UNIVERSITÉ DE LA ROCHELLE

ÉCOLE DOCTORALE SCIENCES ET INGÉNIERIE
POUR L'INFORMATION, MATHÉMATIQUES

Année : 2015

Thèse de Doctorat

Spécialité : Informatique et Applications

Vers un système omni-langage de recherche de mots dans des bases de documents écrits homogènes

Présentée et soutenue par :

Quang Anh BUI

Le 28 septembre 2015

Jury :

Président : Jean Philippe DOMENGER. Professeur à l'université de Bordeaux

Rapporteur : Véronique EGLIN. Professeur à l'INSA de Lyon

Rapporteur : Nicole VICENT. Professeur à l'université Paris-Descartes

Directeur de thèse : Rémy MULLOT. Professeur à l'université de La Rochelle

Encadrant : Muriel VISANI. Maître de Conférences HDR à l'université de La Rochelle.

TABLE DES MATIÈRES

TABLE DES FIGURES

Chapitre 1 : Introduction

Le patrimoine culturel est l'héritage reçu des générations passées, entretenu dans le présent et qui sera légué aux générations futures. L'importance du patrimoine culturel réside dans la richesse des connaissances et du savoir-faire qu'il transmet d'une génération à une autre. Cette transmission du savoir a une valeur sociale et économique pertinente pour les groupes minoritaires comme pour les groupes sociaux majoritaires à l'intérieur d'un État, et est tout aussi importante pour les pays en développement que pour les pays développés. Par contre, l'ignorance, indifférence, les guerres et les conflits, les interventions humaines, les incendies, les catastrophes naturelles, la dégradation naturelle, *etc.* sont des menaces pour le patrimoine culturel. Si nous perdons le patrimoine culturel, nous perdons une partie de notre identité. Nous devons donc protéger le patrimoine culturel et le diffuser au plus grand public.

Une quantité énorme de la connaissance humaine et du patrimoine culturel est souvent stockée dans des documents anciens, répartis dans le monde entier. Chaque année, un nombre croissant de documents anciens sont numérisés pour être préservés et également pour diffuser ce patrimoine culturel au plus grand nombre. Cette numérisation présente de nombreux avantages du point de vue de l'accessibilité et de la durée de conservation des documents anciens. Elle offre en outre aux historiens et autres chercheurs des nouvelles possibilités d'indexation et de recherche associées à leur discipline.

Il existe différentes méthodes pour l'indexation et la recherche d'information dans les documents numérisés. La plus simple d'entre elles est la transcription manuelle sous forme de documents textuels. Le résultat de la recherche est de très bonne qualité. Par contre, le coût de la transcription manuelle est important (par exemple, sur le site http://sandrine-chabre.com/, le tarif pour une transcription manuelle d'un document français ancien est de 5€ à 10€/page, selon l'état de conservation ou la qualité de la numérisation. Le coût de transcription peut s'élever à 10 fois, voire 100 fois, pour les documents de langage rare ou peu connue. Pour éviter cette transcription manuelle, des méthodes de transcription automatique sont disponibles. Il suffit alors d'appliquer un système de reconnaissance de caractères pour convertir les documents originaux en

documents textes. L'information peut alors être recherchée à partir du texte transcrit. Cependant, la qualité médiocre des documents anciens et/ou l'utilisation d'un langage ancien ou rare rend la reconnaissance de caractères difficiles, d'autant que certains alphabets ne disposent d'aucuns systèmes de reconnaissance automatique.

Une alternative aux approches basées sur la reconnaissance de mots est la recherche de mots basés sur le « word spotting ». L'intérêt de ces approches réside dans le fait qu'elles rendent possibles la recherche de mots dans les documents sans avoir besoin d'une transcription (automatique ou manuelle). Nous pouvons définir une taxonomie de ces approches qui distingue deux familles : les approches basées sur une requête image et les approches basées sur une requête textuelle. Dans les approches basées sur une requête image, l'utilisateur fournit une image de mot comme requête et le système cherche les images de mot dans le document qui sont similaires à l'image requête. Ces approches sont similaires aux systèmes de recherche d'images par le contenu (CBIR : Content-Based Image Retrieval). Elles ne demandent pas de connaissances préalables de documents. Ces systèmes sont donc adaptés à des documents de n'importe quel langage, anciens ou modernes, manuscrits ou imprimés. Cependant, l'inconvénient de ces approches est que l'utilisateur doit trouver une occurrence du mot à rechercher, ce qui peut s'avérer fastidieux, voire parfois impossible. De plus, dans le cas où l'utilisateur voudrait juste savoir si un mot spécifique est apparu dans le document ou non, sans en avoir préalablement une occurrence, ces approches deviennent inadaptées. Pour circonscrire ces difficultés, les approches basées sur une requête textuelle sont proposées. Ces approches permettent à l'utilisateur de fournir sa requête texte au lieu de trouver dans le document une occurrence du mot à rechercher. L'utilisateur peut taper dans le clavier le mot à rechercher, ou choisir dans une liste des mots prédéfinis. Même si ces approches sont plus efficaces et conviviales que les approches basées sur une requête image, elles demandent plus ou moins une connaissance préalable du document (par exemple : la transcription, le langage, *etc.*) ce qui n'est pas facile à obtenir, particulièrement dans le cas de documents anciens ou de langage rare.

L'objectif de notre thèse est de définir une nouvelle signature structurelle des images de mots sans connaissance *a priori* du document. Cette nouvelle signature

est basée sur les invariants extraits à partir de la collection de documents (les invariants sont les formes les plus fréquentes dans la collection de document). La requête composée par l'utilisateur et les images de mots dans la collection de documents peuvent être représentées par cette nouvelle signature ceci afin de faire la recherche de mots.

Afin de contourner les inconvénients des systèmes existants dans la littérature, nous proposons un système générique, omni langage et interactif de recherche de mots dans des collections de documents. Nous nous plaçons dans le contexte où le contenu du document est homogène (ce qui est le cas pour les documents anciens où l'écriture est souvent bien soignée et mono-scripteur) et la connaissance préalable du document (le langage, le scripteur, le type d'écriture, le tampon, *etc.*) n'est pas connue. Notre système original est basé sur l'extraction semi-automatique de formes récurrentes dans le texte, appelées « invariants », cette extraction se faisant en interaction avec l'utilisateur. Pour formuler sa requête, l'utilisateur utilisera ces invariants pour la de construction de l'écriture. Les invariants sont aussi utilisés dans la construction des signatures structurelles pour la recherche de mots.

Cette thèse est organisée comme suit : dans le chapitre 2, nous présentons quelques-uns des systèmes de recherche de mots existants dans la littérature, leurs avantages et inconvénients. Puis nous présentons le système de recherche de mots que nous proposons. Dans le chapitre 3, nous présentons le système d'extraction d'invariants à partir de la collection de documents. Dans le chapitre 4, nous présentons notre système de recherche de mots utilisant les signatures structurelles basées sur les invariants extraits. Le chapitre 4 présente la méthode d'évaluation de ces invariants. Nous présentons la composition interactive de requêtes dans le chapitre 5.

Chapitre 2 : Systèmes de recherche de mots dans des collections de documents

I start where the last man left off

Thomas A. Edison

La recherche de mots pour la navigation dans des collections de documents textuels numérisés est un sujet de recherche actif dans la communauté internationale. Il existe différentes méthodes pour l'indexation et la recherche d'informations depuis ce type de documents. Dans ce chapitre, nous présentons différents types de systèmes existants, leurs avantages et inconvénients.

2.1. État de l'art des systèmes de recherche de mots basés sur la reconnaissance de mots (systèmes de transcription automatique)

Les systèmes basés sur la reconnaissance de mots sont les systèmes les plus traditionnels pour la navigation dans des collections de documents numérisés. Il s'agit de reconnaître les mots dans un document numérisé et de les transcrire en texte. L'information peut alors être recherchée à partir du texte transcrit.

La reconnaissance de mots est étudiée depuis longtemps et elle a inspiré beaucoup de travaux de recherche. De nombreuses méthodes ont été proposées ; elles peuvent être regroupées en deux catégories principales : les méthodes de reconnaissances des mots imprimés et les méthodes de reconnaissance des mots manuscrits.

Les systèmes de reconnaissance de mots imprimés sont nombreux et il existe des logiciels (commerciaux / non-commerciaux) connus et communément désignés par le terme OCR (Optical Character Recognition) : Expervision TypeReader, Microsoft Office Document Imaging, Simple OCR, OCROPUS, Abbyy FineReader, *etc.* Avec des documents imprimés modernes, ces logiciels peuvent atteindre de très bons taux de reconnaissance. Des études sur l'état de l'art des systèmes de reconnaissance de mots imprimés peuvent être retrouvées dans les références [Impedovo *et al.* 1991; Rice *et al.* 1992].

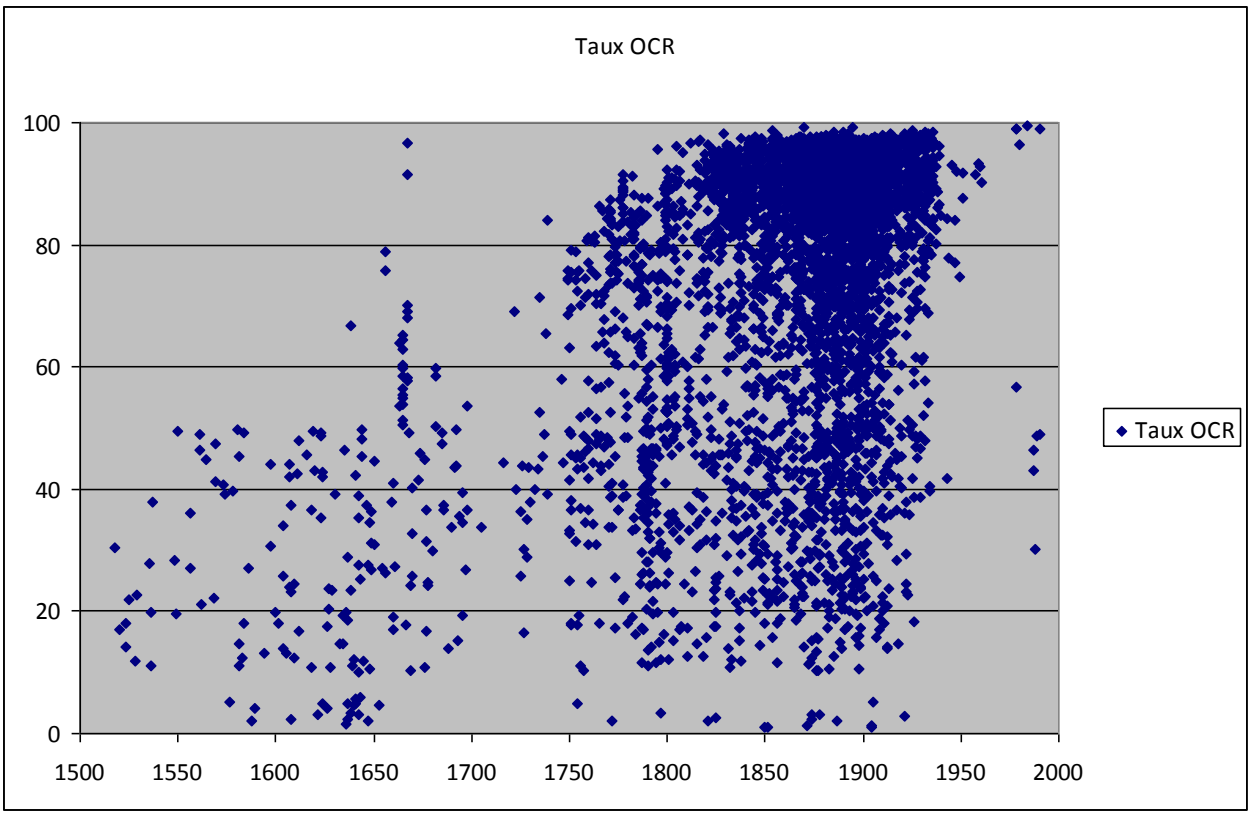


Figure 2.1 : Les performances des systèmes de reconnaissances par rapport à l'époque d'impression des documents [Cron 2012]

Par contre, les taux de reconnaissance des systèmes OCR existants décroissent en présence des documents anciens ou dégradés. Pour illustrer ce propos, la Figure 2.1 montre les performances des systèmes de reconnaissance de caractères imprimés par rapport à la date d'impression des documents. Ce diagramme est extrait d'un rapport interne de la Bibliothèque Nationale de France (BNF) [Cron 2012]. Il montre que les systèmes de reconnaissance peuvent atteindre des performances très variables suivant la datation. En particulier, il montre qu'il n'est pas envisageable d'utiliser ce type d'algorithme pour des documents datant avant 1750.

Les systèmes de reconnaissance de mots manuscrits : La reconnaissance des mots manuscrits est également un sujet de recherche actif depuis plusieurs décennies. Il y a principalement deux types de systèmes de reconnaissance de mots manuscrits : les systèmes de reconnaissance de mots hors-ligne et les systèmes de reconnaissance de mots en-ligne. Dans le contexte de la transcription automatique de documents numérisés, seuls les systèmes de reconnaissance des mots hors-ligne sont utilisés. Les systèmes de reconnaissance de mots manuscrits hors-ligne ont en général des performances inférieures à celles de la reconnaissance de mots imprimés. Ceci est dû en particulier aux difficultés liées aux variations de

l'écriture dépendant du scripteur et, le cas échéant, du caractère cursif de l'écriture. De ce point de vue, concernant les systèmes de reconnaissance de mots manuscrits, il est assez naturel de dissocier la reconnaissance de caractères isolés et la reconnaissance de mots manuscrits cursifs. Les avancées dans la reconnaissance des caractères manuscrits isolés réalisées au cours des dernières années sont remarquables. Dans la littérature, les taux de reconnaissance atteignent 99,5% dans [Suen *et al.* 1993]. Par contre, la reconnaissance des mots manuscrits cursifs reste une problématique ouverte dès lors que le vocabulaire est large. Classiquement, on distingue deux types d'approches pour la reconnaissance des mots cursifs dans la littérature : les approches globales (les plus anciennes), et les approches analytiques.

Les méthodes globales [Simon & Baret 1992; Senior & Fallside 1993] considèrent chaque mot comme une forme globale à reconnaître. Plus précisément, une segmentation explicite est appliquée pour segmenter les images de mots dans le document. Puis, les caractéristiques des images de mots sont extraites. Par exemple, [Simon & Baret 1992] extraient des chaînes appelées « chaînes descriptives » et qui représentent les caractéristiques structurelles d'une image de mot. [Senior & Fallside 1993] proposent une approche basée sur un réseau de neurones. Les auteurs utilisent alors les directions quantifiées des *strokes* dans l'image de mot comme caractéristiques. Un inconvénient majeur des méthodes globales est qu'elles nécessitent une grande base d'apprentissage contenant tous les mots dans le lexique ce qui est difficile à obtenir. Il est classique de dire que les méthodes globales ne sont donc pas adaptées aux applications de grand lexique, surtout quand une variabilité forte est observée.

Pour contourner ce problème, les approches analytiques ont été développées : [Hu *et al.* 1996; Hu *et al.* 2000; Plamondon & Srihari 2000; Kavallieratou *et al.* 2002; Bianne-Bernard *et al.* 2011; Liwicki & Bunke 2006; Graves *et al.* 2008; España-Boquera *et al.* 2011]. Ces approches se basent généralement sur une reconnaissance de l'enchaînement des caractères ou des morceaux de caractères (graphèmes) composante les mots. Dans les méthodes analytiques basées sur une segmentation explicite : [Plamondon & Srihari 2000; Kavallieratou *et al.* 2002], le mot est d'abord découpé en caractères, puis, les caractères segmentés sont reconnus par un ensemble des modèles de caractères couvrant tous les caractères possibles de la langue considérée. Les approches analytiques sont donc adaptées aux applications de grand lexique. Le principal désavantage de ces méthodes est que le résultat de la reconnaissance dépend de la segmentation, ce qui n'est en général pas fiable dans les cas des documents non-contraints (les documents multi-scripteurs ou les documents où les écritures sont peu soignées).

Pour contourner ce problème, des méthodes analytiques basées sur une segmentation implicite et un alignement des modèles de caractères par modèles de Markov cachés (MMC) ont été proposées [Hu *et al.* 1996; Hu *et al.* 2000; Liwicki & Bunke 2006; Bianne-Bernard *et al.* 2011]. Dans ces approches, le problème de segmentation est résolu pendant l'étape de reconnaissance. Plus précisément, les mots d'entrées sont d'abord sur-segmentés en graphèmes. Puis, le MMC se charge du décodage des mots par alignement des modèles de caractères conditionnellement à l'apparence des graphèmes observés.

Toutefois, les modèles de Markov cachés ont quelques inconvénients. Un de ces inconvénients est qu'ils supposent que la probabilité de chaque observation ne dépend que de l'état actuel, ce qui rend les effets contextuels difficiles à modéliser. Pour contourner ces limitations des modèles de Markov cachés, des méthodes incluant des classifieurs discriminants [España-Boquera *et al.* 2011; Graves *et al.* 2008] (souvent sous forme de systèmes hybrides en combinaison avec les modèles de Markov cachés) ont été proposées , afin de résoudre le problème de segmentation.

Conclusion :

Malgré le fait que plusieurs méthodes aient été proposées, les taux de reconnaissance des méthodes existantes ne sont en général pas suffisants pour les systèmes de recherche de mots, en particulier dans des documents anciens ou dégradés et dans le cas de vocabulaire large. Lorsque les documents sont anciens ou dégradés, une difficulté de l'approche basée sur la reconnaissance de mots est le prétraitement du document, surtout le nettoyage des bruits et l'amélioration de la qualité de l'image.

Une autre limitation des systèmes basés sur la reconnaissance de mots est qu'ils sont généralement dédiés à des langages ou des types d'écritures particuliers. Par exemple, un système dédié pour le français ne peut pas fonctionner correctement sur les documents écrits en chinois. Cela devient problématique en particulier dans le cas des documents anciens dont le script est rare.

En conclusion, à cause des limitations présentées ci-dessus, les systèmes basés sur la reconnaissance de mots ne sont pas adaptés dans notre contexte présenté dans le chapitre 1. Une alternative aux approches basées sur la reconnaissance de mots est les systèmes de recherche de mots basés sur le « word spotting ». Nous présentons ces approches dans la section suivante (section 2.2).

2.2. État de l'art des systèmes de recherche de mots basés sur le « word spotting »

L'intérêt principal des approches basées sur le « word spotting » réside dans le fait que ces approches rendent possibles de faire la recherche de mots dans les documents (imprimés ou manuscrits) sans avoir besoin de la transcription (coûteuse dans le cas de la transcription manuelle, ou peu fiable avec les documents manuscrits ou anciens dans le cas de la transcription automatique).

Nous pouvons définir une taxonomie des approches de word spotting qui distingue deux familles :

- *Le word spotting basé sur une requête image :* Ces approches consistent à retrouver dans les documents, les images de mots similaires à l'image de requête donnée par l'utilisateur
- *Le word spotting basé sur une requête textuelle :* Ces approches permettent à l'utilisateur de fournir sa requête sous la forme de texte pour rechercher dans les documents, les images de mots correspondantes.

Nous présentons dans cette section ces deux familles.

2.2.1. Le word spotting basé sur une requête image

Le word spotting basé sur une requête image ne nécessite pas de connaissance préalable sur les documents. La plupart des approches de cette catégorie (comme les méthodes présentées ci-après) sont dédiées aux mots manuscrits, mais elles peuvent également s'appliquer sur des documents imprimés. Dans ces approches, les caractéristiques des images (l'image de requête et les images dans la base de données) sont extraites. Sur la base de ces caractéristiques, les distances entre l'image de requête et les images dans la base de données sont calculées. La réponse de cette requête est un ensemble d'images de mots dans le document qui sont les plus similaires à l'image de requête. Ce type de méthode se rapproche des systèmes de recherche d'images par le contenu (CBIR : Content Based Image Retrieval).

Les approches de ce type varient essentiellement en fonction des caractéristiques utilisées pour décrire les mots (la projection de profil : [Manmatha *et al.* 1996; Rath & Manmatha 2003] ; l'histogramme du gradient local : [Perronnin *et al.* 2008] ; contexte de forme [Rusinol & Llados 2013], *etc.*) et de la mesure de distance (la distance Euclidienne [Manmatha *et al.* 1996] ; l'algorithme Dynamic Time Wrapping [Rath & Manmatha 2003; Perronnin *et al.* 2008] ; la distance cosinus [Rusinol & Llados 2013], *etc.*). Un état de l'art plus complet et spécifique aux descripteurs est présenté en Annexe.

[Manmatha *et al.* 1996] utilisent la projection de profil comme caractéristiques de l'image. Lorsque ces caractéristiques des images sont extraites, [Manmatha *et al.* 1996] calculent la distance Euclidienne entre les images. Puis, un algorithme de *clustering* est appliqué pour grouper les images similaires dans un même *cluster*. Idéalement, chaque *cluster* devrait contenir toutes les occurrences d'un mot. Les *clusters* qui contiennent les termes significatifs sont sélectionnés et indexés manuellement. Quand l'utilisateur donne sa requête, le système recherche le *cluster* le plus proche de cette image requête et retourne à l'utilisateur toutes les autres images (avec leurs positions dans le document) de ce *cluster*. Comme un raffinement de cette méthode, au lieu d'utiliser la distance Euclidienne, les mêmes auteurs utilisent dans [Rath & Manmatha 2003] l'algorithme Dynamic Time Wrapping (DWT), un algorithme de programmation dynamique. L'avantage de DWT sur la distance Euclidienne, est qu'il trouve une correspondance non-linéaire entre deux formes. Les résultats utilisant DWT sont ainsi de meilleure qualité.

[Rothfeder *et al.* 2003] extraient les points saillants des images et font la mise en correspondance entre les points saillants de deux images pour calculer la similarité entre elles. Les points saillants d'une image sont extraits par un détecteur de Harris. La mise en correspondance entre les points est établie en utilisant la mesure SSD (Sum of Squared Differences) entre leurs coordonnées. La similarité entre deux images est la somme des distances entre les vecteurs de caractéristiques des points saillants correspondant à ces deux images.

[Wang 2014] utilisent un graphe comme descripteur des images de mots. Plus précisément, les auteurs utilisent une méthode de squelettisation pour obtenir le squelette des images de mots. Ensuite, ils trouvent les points saillants dans le squelette. Les points saillants sont les points de croisement, les points d'extrémité et les points de courbure élevée. Après l'extraction des points saillants, les auteurs représentent l'image de mot comme un graphe $G = (V, E, \mu, \nu)$ où V est l'ensemble des nœuds du graphe correspondant aux points saillants, E est l'ensemble des arêtes, correspondant aux branches entre les points saillants (dans le squelette de l'image), μ est la fonction d'étiquetage des nœuds. Chaque nœud est étiqueté par le descripteur Contexte de Forme [Belongie *et al.* 2002] du point saillant correspondant (le descripteur Contexte de Forme d'un point saillant est calculé en utilisant le contour de l'image). Enfin, ν est la fonction d'étiquetage des arêtes. Chaque arête est étiquetée par la longueur de la branche correspondante. Après la représentation des images de mots en graphes, la distance entre 2 images de mots est la distance entre 2 graphes correspondants. Pour calculer la distance entre 2 graphes, les auteurs utilisent la distance d'édition approximative proposée dans [Riesen & Bunke 2009]. En utilisant le graphe, les informations topologiques et structurelles des écritures sont capturées. Mais l'inconvénient de cette méthode

repose sur la complexité élevée du calcul de la distance d'édition entre les graphes. De plus, cette méthode dépend fortement sur la squelettisation qui n'est pas stable vis-à-vis de potentielles dégradation du document. Cela peut entraîner la déformation de la représentation du graphe.

Les méthodes proposées par [Rath & Manmatha 2003; Manmatha *et al.* 1996], [Rothfeder *et al.* 2003] et [Wang 2014] donnent de bons résultats dès lors qu'il existe une forme d'homogénéité de l'écriture. Dans le cas des documents non-contraints et/ou multi-scripteur, la performance de ces méthodes n'est alors pas satisfaisante.

Quand il s'agit de documents dégradés, non-contraints ou multi-scripteur, plusieurs méthodes ont été proposées [Keaton *et al.* 1997; Perronnin *et al.* 2008; Leydier *et al.* 2007; Rusinol & Llados 2013]. Ces méthodes utilisent des caractéristiques qui sont plus robustes à la variation de l'écriture.

[Keaton *et al.* 1997] extraient les caractéristiques de cavité pour calculer la similarité entre 2 images. Les caractéristiques de cavité capturent les variations locales de l'image du mot qui sont utiles pour discriminer les formes similaires. Pour calculer les caractéristiques de cavité, un algorithme morphologique est appliqué en utilisant des combinaisons de dilatations ou d'érosions dans des directions et des intersections différentes. Il y a 6 types de caractéristiques de cavité : Est, Ouest, Nord, Sud, Centre et Trou. Les caractéristiques de cavité sont codées par des modèles de graphe dans lesquels la relation spatiale relative est préservée. Pour la description plus détaillée du calcul et du codage de caractéristiques de cavité, nous renvoyons le lecteur à [Keaton *et al.* 1997]. La similarité entre 2 images est alors calculée en utilisant un algorithme probabiliste d'appariement de graphes basé sur un raisonnement Bayésien. En utilisant les caractéristiques de cavité et le codage basé sur les modèles de graphe, la méthode de [Keaton *et al.* 1997] préserve la relation spatiale entre les éléments constitutifs (*strokes*) du mot. Cette méthode est donc robuste à la variation de l'écriture.

La méthode de [Leydier *et al.* 2007] est basée sur les caractéristiques différentielles qui sont comparés en utilisant une méthode d'appariement élastique cohésive. Plus précisément, [Leydier *et al.* 2007] extraient le gradient des pixels de l'image comme caractéristiques (pour plus de détails sur le calcul du gradient, nous renvoyons le lecteur à [Leydier *et al.* 2007]). La distance entre l'image de requête et l'image dans la base de données est calculée comme la somme de distances entre les gradients des pixels correspondant à ces 2 images. La distance entre 2 gradients des 2 pixels est définie comme la distance angulaire des

2 gradients quand leurs intensités sont toutes inférieures à un seuil prédéfini ϵ (le seuil ϵ est choisi par l'utilisateur en utilisant une interface visuelle). Pour mettre en correspondance les pixels de 2 images, une méthode d'appariement naïve ou une méthode d'appariement élastique naïve peuvent être appliquées. À noter que la complexité de ces méthodes d'appariement est élevée. Pour réduire la complexité, les auteurs proposent une méthode d'appariement élastique cohésive basée sur des zones d'intérêt afin de mettre en correspondance seulement les parties les plus informatives des images (pour plus de détails sur la méthode d'appariement élastique cohésive et la méthode d'extraction des zones d'intérêt, nous renvoyons le lecteur à [Leydier *et al.* 2007]). Grâce à l'utilisation des caractéristiques différentielles et d'une méthode d'appariement élastique cohésive, la méthode de [Leydier *et al.* 2007] est robuste vis-à-vis de variations de l'écriture.

[Perronnin *et al.* 2008] proposent une méthode utilisant des caractéristiques invariantes à la rotation, à la transformation et aux changements d'échelle. Dans cette méthode, le document est segmenté en images de mots. Puis, ces images de mots sont normalisées. Pour chaque image normalisée, les vecteurs de caractéristiques sont calculés en utilisant une fenêtre glissante se déplaçant de gauche à droite de l'image. À chaque position de cette fenêtre, des caractéristiques incluant le filtre Gaussien et l'histogramme du gradient local sont calculées. Lorsque l'utilisateur donne sa requête, le système retourne les images de mots (et leur position dans le document) pour lesquelles la distance DTW avec l'image requête est inférieure à un seuil fixé. Pour tester leur méthode, les auteurs de [Perronnin *et al.* 2008] ont appliqué leur méthode sur 630 lettres manuscrites numérisées (écrites en français), reçues par le service client d'une compagnie. Chaque lettre est écrite par un même scripteur. La performance de cette méthode est donc logiquement plus élevée que celles des méthodes Manmatha *et al.* [Manmatha *et al.* 1996; Rath & Manmatha 2003] et de Rothfeder *et al.* [Rothfeder *et al.* 2003].

En fin, l'approche de [Rusinol & Llados 2013] emploie la méthode dite « sacs de mot » pour l'indexation et la recherche. Les auteurs extraient, à partir de chaque image dans la base de données, les descripteurs SIFT ou les descripteurs « contextes de forme ». Une fois que les descripteurs sont calculés, les auteurs appliquent la méthode de *clustering* K-moyennes pour obtenir un dictionnaire qui quantifie les descripteurs en mots visuels. Chaque descripteur est associé à un mot visuel représentant le *cluster* de ce descripteur. Chaque image dans la base de données est ainsi projetée dans l'espace de mots visuels. Dans la phase de recherche, lorsque l'utilisateur fournit l'image de requête, les descripteurs (SIFT ou contexte de forme) sont calculés. Puis, l'image de requête

est projetée dans l'espace de mots visuels. Ensuite, les distances cosinus entre l'image de requête et les images dans la base de données sont calculées. Le système retourne les images les plus proches de l'image de requête. Pour améliorer la performance du système, [Rusinol & Llados 2013] appliquent la technique de retour de pertinence. À chaque itération de recherche, l'utilisateur est sollicité pour retourner son expertise sur les résultats : quels résultats sont corrects et quels résultats ne sont pas corrects. À partir du retour de l'utilisateur, la formule de Rocchio [Rocchio 1971] est appliquée pour raffiner le résultat de recherche. Grâce en particulier aux interactions avec l'utilisateur, la méthode de [Rusinol & Llados 2013] est adaptée dans le cas où l'apparence de l'écriture est variable.

Conclusion :

Nous avons présenté dans cette section quelques-unes des approches de word spotting basées sur une requête image. Ces approches n'exigent aucune connaissance préalable des documents. Les systèmes basés sur ces approches peuvent être appliqués sur les écrits en n'importe quel langage, des documents imprimés ou des documents manuscrits.

La limitation principale des approches basées sur une requête image est que l'utilisateur doit trouver une occurrence (voire plusieurs afin d'améliorer le résultat) du mot à rechercher dans le document, ce qui peut s'avérer fastidieux voir parfois impossible. De plus, dans le cas où l'utilisateur voudrait juste savoir si un mot spécifique est apparu dans la collection de documents ou non, sans en avoir préalablement une occurrence, le système de recherche devient inadapté. Il est beaucoup plus facile pour l'utilisateur de taper sur le clavier le mot qu'il veut rechercher. À partir de cette idée, des approches basées sur des requêtes textuelles ont été proposées. Nous présentons ces approches dans la section suivante (section 2.2.2).

2.2.2. Le word spotting basé sur une requête textuelle

Les approches basées sur une requête textuelle permettent à l'utilisateur de fournir sa requête sur la base de mots composés textuellement (au lieu de rechercher dans le document un exemple du mot), afin de localiser dans les documents les images de mots correspondantes. L'utilisateur peut taper sur le clavier le mot-clé qu'il veut rechercher, ou choisir dans une liste des mots prédéfinie. Dans la plupart des approches présentées ci-après, comme par exemple

les méthodes [Marinai *et al.* 2006; Konidaris *et al.* 2007; Leydier *et al.* 2009; Liang *et al.* 2012; Rodríguez-Serrano & Perronnin 2009; Aldavert *et al.* 2013], les images de mots sont explicitement segmentées à partir des documents. Ces méthodes retournent les images de mots (segmentées et stockées dans la base de données) qui correspondent au mot-clé recherché. D'autres approches comme la méthode [Fischer *et al.* 2012], segmente les lignes de mots au lieu de segmenter les images de mots, et retourne la ligne qui contient le mot-clé avec la position de la zone qui correspond au mot-clé dans la ligne de mots.

La plupart de ces approches reposent sur la construction des modèles de mots, de caractères ou de graphèmes issus de la collection de documents. Ces modèles sont construits ***manuellement*** [Konidaris *et al.* 2007; Leydier *et al.* 2009; Marinai *et al.* 2006] ou ***par apprentissage automatique*** [Fischer *et al.* 2012; Manmatha *et al.* 2012; Rodríguez-Serrano & Perronnin 2009; Liang *et al.* 2012; Aldavert *et al.* 2013]. Dans la phase de recherche, lorsque l'utilisateur fournit la requête texte, ces modèles sont assemblés afin de former la requête. Certaines approches, comme les méthodes [Konidaris *et al.* 2007; Leydier *et al.* 2009; Marinai *et al.* 2006] présentées ci-après ont été conçues spécifiquement pour des documents imprimés tandis que d'autres [Leydier *et al.* 2009], [Manmatha *et al.* 2012; Liang *et al.* 2012; Rodríguez-Serrano & Perronnin 2009; Fischer *et al.* 2012], conçues pour les documents manuscrits, peuvent également être appliquées dans le cas imprimé.

2.2.2.1. Méthodes sans apprentissage automatique

Dans les méthodes [Konidaris *et al.* 2007; Leydier *et al.* 2009; Marinai *et al.* 2006], les modèles de caractères (ou graphèmes) sont construits ***manuellement*** par extraction des imagettes correspondantes dans le document. Dans la phase de recherche, ces modèles de caractères (ou de graphèmes) sont alignés pour générer synthétiquement une image de requête. Le système prend cette image de requête synthétisée pour rechercher les images similaires dans la base de données. Ces approches sont similaires aux approches basées sur une requête image (présentées dans la section 2.2.1), sauf qu'ici l'image de requête est générée synthétiquement.

[Marinai *et al.* 2006] génèrent l'image de requête à partir de la requête texte en utilisant une police spécifique. Cette méthode est rapide, efficace et conviviale, mais elle ne peut s'appliquer qu'à des documents imprimés. De plus, elle nécessite une connaissance *à priori* sur la police du document traité.

Dans la méthode de [Konidaris *et al.* 2007], chaque caractère texte (code ASCII) est associé manuellement à une image de caractère. Lorsque l'utilisateur tape sa requête sur le clavier, l'image de requête est construite par la concaténation des images correspondantes aux caractères texte. Pour que l'image de requête soit plus

réelle, les auteurs détectent la ligne de base de chaque image de caractère, et dans la phase de synthèse, les images de caractères sont alignées sur cette ligne de base.

Dans la méthode de [Leydier *et al.* 2009], les auteurs extraient les modèles de graphèmes (morceaux de caractères) plutôt que le modèle de caractères. Plus précisément, les auteurs de [Leydier *et al.* 2009] construisent manuellement un « livre de glyphes » et une « grammaire » spécifique à chaque document:

- Le « livre de glyphes » contient les images de graphèmes et leurs relations spatiales.
- La « grammaire » définit un ensemble des règles d'édition permettant d'éditer l'image de requête à partir du texte donné, en utilisant le livre de glyphes.

Lorsque l'utilisateur fourni sa requête textuelle, l'image de requête est générée en utilisant le livre de glyphes et la grammaire.

Bien que les méthodes [Konidaris *et al.* 2007; Leydier *et al.* 2009; Marinai *et al.* 2006] soient efficaces et conviviales, elles peuvent seulement être appliquées aux contextes où les écritures sont homogènes, par exemple les documents imprimés [Konidaris *et al.* 2007; Marinai *et al.* 2006] ou les documents manuscrits anciens où les écritures sont bien soignées et écrites par un seul scripteur [Leydier *et al.* 2009]. Mais, leur principal défaut est qu'elles reposent sur une intervention manuelle de l'utilisateur qui peut se révéler fastidieuse. Pour contourner les difficultés des documents non homogènes (par exemple, les documents manuscrits dégradés, multi-scripteur, et/ou non contraints), des approches basées sur ***un apprentissage automatique*** ont été proposées [Fischer *et al.* 2012; Manmatha *et al.* 2012; Rodríguez-Serrano & Perronnin 2009; Liang *et al.* 2012; Aldavert *et al.* 2013]. La plupart de ces systèmes reposent sur la construction par apprentissage de modèle de mots ou de modèles de caractères. Ils sont donc proches des méthodes de transcription, sauf qu'ici la reconnaissance se fait de manière ciblée pour une requête donnée de l'utilisateur.

2.2.2.2. Méthodes avec apprentissage automatique

L'objectif de ces méthodes est de construire les modèles de mots, de caractères ou de graphèmes par apprentissage automatique. Dans la phase de recherche, lorsque l'utilisateur fournit sa requête textuelle, ces modèles sont assemblés afin de former la requête. [Rodríguez-Serrano & Perronnin 2009] construisent les modèles de mots en utilisant un apprentissage supervisé de MMC (Modèles de Markov Cachés) [Welch 2003]. Plus précisément, dans la phase d'apprentissage, les auteurs utilisent une base d'apprentissage qui contient les images de mots

organisées en classes. Un lexique est donc défini comme une liste des mots dans la base d'apprentissage. Pour construire les modèles de mots, l'histogramme du gradient local [Perronnin *et al.* 2008] est extrait de chaque image dans la base d'apprentissage. Ensuite, pour chaque mot M dans le lexique, un modèle de MMC de ce mot est construit en utilisant un nombre fixé d'états (ici 10) et l'algorithme de Baum – Welch [Welch 2003]. Dans la phase de recherche, l'utilisateur choisit dans le lexique le mot qu'il veut rechercher. Le modèle Q de la requête est donc chargé. Pour chaque image de mot X (pré-segmentée) dans le document, le système calcule son score qui est la probabilité *a posteriori* du modèle Q sachant l'image X : $P(Q|X)$ en utilisant l'algorithme de Viterbi [Welch 2003]. Le système retourne les images de mot dont le score est le plus élevé. La méthode [Rodríguez-Serrano & Perronnin 2009] est rapide avec un taux de reconnaissance élevée. Mais, cette méthode ne permet pas à l'utilisateur de rechercher les mots qui ne sont pas dans le lexique. C'est l'inconvénient majeur de cette méthode et la rend peu flexible. De plus, pour l'application dans les documents de grand lexique, ces méthodes reposent sur une grande base d'apprentissage contenant tous les mots dans le lexique ce qui est difficile à obtenir.

Pour être indépendant du lexique et donc rendre le système plus flexible, [Liang *et al.* 2012] construisent les modèles de caractères basés sur les graphèmes. La requête est une concaténation des modèles de caractères, basée sur le texte fournit par l'utilisateur. Plus précisément, dans la phase d'apprentissage, les auteurs segmentent les mots en graphèmes à partir d'une base d'apprentissage. Les graphèmes extraits sont ensuite groupés par la méthode de *clustering* SOM (Self Organizing Map). Les *clusters* de graphèmes sont utilisés pour construire les modèles de caractères. De façon plus précise, en utilisant l'annotation associée à la base d'apprentissage, chaque modèle de caractère c est représenté par un vecteur $(p_{1_c}, p_{2_c}, \dots, p_{N_c})$. L'élément p_{i_c} $(i = 1 \dots N)$ du vecteur est la probabilité de l'un des graphèmes du *cluster* i de se présenter dans une instance (une image) du caractère c. Lorsque l'utilisateur fournit sa requête texte, le modèle de requête est construit par la concaténation des vecteurs des caractères composant le mot. Dans la phase de recherche, les images de caractères sont extraites à partir de chaque image de mots dans la base de données en utilisant la méthode de segmentation proposées par les auteurs. Étant donné la requête Q et l'image de mot X (pré-segmentée) dans la base de données, les auteurs calculent, pour chaque $k^{ième}$ caractère de l'image de mot X, la probabilité d'être une instance du $k^{ième}$ caractère de Q : γ_{Q_k,X_k}. La distance entre l'image de mot X et la requête Q est : $dist(X,Q) = \sum_{k=1}^{M} \gamma_{Q_k,X_k}$ où M est le nombre de caractères de la requête Q. Pour calculer γ_{Q_k,X_k}, Liang *et al.* utilisent la segmentation de l'image X en graphèmes. Supposons que j soit un graphème du $k^{ième}$ caractère de l'image de mot X (X_k), les auteurs calculent

la probabilité du graphème j de se présenter dans une instance du $k^{ième}$ caractère de la requête Q (Q_k) : $p(j, Q_k)$. La possibilité du $k^{ième}$ caractère de l'image de mot X d'être une instance du $k^{ième}$ caractère de Q : γ_{Q_k,X_k} est donc : $\gamma_{Q_k,X_k} = \sum_{j \in X_k} p(j, Q_k)$. Pour calculer $p(j, Q_k)$, le graphème j est associé à un *cluster* de graphèmes en utilisant la méthode SOM et la carte de Kohonen construit dans la phase d'apprentissage. Pour plus de détails, nous renvoyons le lecteur à [Liang *et al.* 2012]. En utilisant l'apprentissage pour construire les modèles de caractères basés sur les graphèmes, la méthode de Liang *et al.* donne de bons résultats dans les cas des documents homogènes et non-homogènes. Mais elle dépend à la segmentation explicite des mots en caractères, qui n'est pas fiable dans le cas des documents dégradés (qui contiennent des bruits, des dégradations et des distorsions).

Afin d'éviter la segmentation explicite des mots en caractères dans la phase d'apprentissage, [Fischer *et al.* 2012] construisent les modèles de caractères en utilisant des MMC [Welch 2003]. Dans ce système, les images de la ligne de mots du document sont stockées dans la base de données. Lorsque la requête (le mot-clé) est fournie par l'utilisateur, le modèle de la requête est construit par la concaténation des modèles de caractères. Puis, un score entre le modèle de la requête et une image de la ligne de mot est calculé. La base d'apprentissage contient les images de lignes de mots avec la transcription. Dans la phase d'apprentissage, les caractéristiques des images de ligne de mots sont extraites. Chaque caractère texte est attribué à un modèle MMC. Après l'extraction de caractéristiques des images dans la base d'apprentissage, les modèles MMC des lignes de mots sont construits comme la concaténation des modèles de caractères selon la transcription. Lorsque que les modèles de ligne de mots sont entraînés (en utilisant l'algorithme de Baum – Welch [Welch 2003]), les modèles de caractères sont aussi entraînés. Dans la phase de recherche, lorsque l'utilisateur fournit son mot de requête texte, le modèle Q de la requête est construit par la concaténation des modèles de caractères correspondants. Pour chaque ligne X dans la collection de documents, le système calcule le score de la ligne : $s(X, Q)$. Le calcul de ce score est basé sur le calcul des probabilités : $P(Q|X_{a,b})$ ($X_{a,b}$ est la partie dans la ligne X, définie par deux positions a et b), en utilisant l'algorithme de Viterbi [Welch 2003]. Si ce score est supérieur à un certain seuil, l'image de la ligne de mots est retournée avec la position du mot-clé dans l'image de la ligne. La description complète de la méthode est proposée dans [Fischer *et al.* 2012]. La méthode proposée par [Fischer *et al.* 2012] construit les modèles de caractères sans segmentation explicite en caractères ce qui est un atout important en particulier dans le cas de documents dégradés. Elle est de plus indépendante du lexique, ce qui rend le système flexible et fiable. La limitation de cette méthode porte sur la nécessité de connaître

préalablement le langage des documents. De plus, la complexité de la phase de recherche est élevée.

Ne reposant pas sur la construction de modèles (ni de mots, ni de caractères), la méthode de [Aldavert *et al.* 2013] combine la représentation textuelle et la représentation visuelle des images de mots dans un espace commun en utilisant l'algorithme d'apprentissage ASL (Analyse Sémantique Latente). Grâce à la projection dans un espace commun, l'utilisateur peut utiliser soit la requête image, soit la requête textuelle pour faire la recherche des mots dans les documents qui sont normalement représentés par des descripteurs visuels. Plus précisément, les auteurs construisent un espace commun et calculent la matrice de transformation pour transformer la représentation textuelle ou la représentation visuelle vers l'espace commun. La représentation textuelle est formulée en termes de descripteurs de caractères n-gram tandis que la représentation visuelle est basée sur le schéma de « sac de mot visuels » en utilisant des caractéristiques de gradient. Idéalement, les représentations textuelles et visuelles d'un même mot se trouvent sur un même point dans cet espace commun. La matrice de transformation est calculée en utilisant une base d'apprentissage et l'algorithme ASL (Analyse Sémantique Latente). La base d'apprentissage contient les images de mots avec leur transcription. Après la construction de l'espace commun, les vecteurs de caractéristiques (les représentations visuelles des mots) des images de mots dans la collection de documents sont calculés et sont projetés dans l'espace commun. Dans la phase de recherche, lorsque l'utilisateur donne sa requête sous forme de texte, les représentations textuelles (les descripteurs de caractère n-gram) de la requête sont calculées et projetées dans l'espace commun. La distance cosinus entre la requête et les images de mots dans la base de test est calculée dans l'espace commun. Le système retourne les images des mots les plus similaires à la requête selon ses représentations textuelles et visuelles. Les auteurs de [Aldavert *et al.* 2013] ont testé leur méthode en utilisant la base George Washington qui contient 4864 images de mots segmentés. Pour les requêtes des mots qui sont dans le lexique (les mots de la transcription), le taux de reconnaissance est de 76,2%. Pour les requêtes de tous les mots (dans le lexique et hors du lexique), le taux de reconnaissance est de 56,54%.

Conclusion :

Nous avons présenté dans cette section quelques-unes des approches basées sur une requête textuelle. Ces approches permettant à l'utilisateur de fournir sa requête par la composition d'un texte ce qui les rend efficaces et conviviales et surtout en s'affranchissant des occurrences. Parmi les approches de ce type, celles qui reposent sur la génération synthétique de l'image de requête donnent de bons

résultats uniquement avec les documents de contenu homogène (les documents imprimés, les documents manuscrits dont les écritures sont homogènes et écrites par un seul scripteur, *etc.*), tandis que les approches basées sur l'apprentissage donnent de bons résultats même avec des documents non homogènes. Généralement, la performance des approches avec apprentissage surpasse la performance des méthodes basées sur une requête image.

La limitation principale de ce type d'approches basées sur des requêtes textuelles est cependant la nécessité d'une connaissance préalable de la collection de documents (transcription, police ou style de l'écriture, langage des documents, règles d'édition, alphabet, *etc.*), ce qui n'est pas forcément facile à obtenir, en particulier pour les documents anciens ou quand le langage utilisé est rare.

2.3. Synthèse des méthodes existantes de recherche des mots

	Convivialité du requêtage	**Performance avec documents imprimés**		**Performance avec documents manuscrits**		**Nécessite une connaissance préalable du document ?**
		Modernes	**Anciens**	**Multi-scripteurs**	**Anciens**	
Transcription automatique	**Bon**	**Très bon**	Moyenne	Mauvais – Moyenne	Mauvais - Moyenne	Oui
Word spotting basé sur une requête image	Mauvais	**Très bon**	**Bon**	**Moyenne - Bon**	**Moyenne - Bon**	**Non**
Word spotting basé sur une requête textuelle, sans apprentissage	**Bon**	**Très bon**	**Bon**	Mauvais - Moyenne	**Moyenne - Bon**	Oui
Word spotting basé sur une requête textuelle avec apprentissage	**Bon**	**Très bon**	**Bon**	**Moyenne - Bon**	**Moyenne - Bon**	Oui

Tableau 2.1 : Résumé des systèmes de recherche des mots

Le Tableau 2.1 montre un résumé des systèmes de recherche de mots. Ce tableau est naturellement lié à l'état actuel de la littérature. Selon ce tableau, lorsque nous voulons construire un système de recherche des mots pour des documents multi-langage, homogènes et non-contraints, sans connaissance préalable du document, nous pouvons utiliser les approches de word spotting basé sur une requête image. Lorsque nous voulons construire un système de recherche de mots plus intuitifs pour des documents homogènes ainsi que des documents non-contraints, nous pouvons utiliser des approches de word spotting basé sur une requête textuelle avec apprentissage.

2.4. Système omni-langage de navigation sur la collection de documents, basé sur l'extraction d'invariants et la composition de requête

Afin de contourner les désavantages des systèmes existants, nous proposons un système générique, omni langage et interactif de recherche de mots dans des collections de documents, fonctionnant même si l'alphabet n'est pas connu. Notre système doit pouvoir fonctionner sur des documents multi-langage, anciens ou modernes, manuscrits ou imprimés. Nous nous plaçons dans le contexte où le contenu du document est homogène. C'est-à-dire, dans le cas de documents manuscrits, que les documents soient écrits par un même scripteur et que les écritures soient bien soignées, et dans le cas des documents imprimés, qu'une seule police soit utilisée pour les documents.

Notre système ne peut pas être basé sur la transcription automatique à cause de la performance médiocre de systèmes dans le cas des documents anciens. Notre système ne peut pas non plus être basé sur le word spotting par requête textuelle car de tels systèmes nécessitent une reconnaissance préalable sur le document. Notre système ne peut pas non plus être basé sur une requête image, car nous souhaitons un mode de requêtage plus convivial pour l'utilisateur. Nous avons donc conçu un système original basé sur l'extraction semi-automatique de formes récurrentes dans le texte (appelés « invariants »), menée en interaction avec l'utilisateur. Pour formuler sa requête, l'utilisateur utilisera ces invariants véritables de construction de l'écriture. La recherche des mots correspondants à la requête de l'utilisateur se fera à partir d'une signature structurelle calculée à partir de l'apparence et de l'agencement spatial des invariants. Plus précisément, notre système repose sur trois étapes :

- ***Étape 1*** : ***Extraction d'invariants***

Les invariants sont les formes les plus fréquentes dans la collection de document. Il peut s'agir par exemple de caractères dans le cas des langages basés sur un alphabet, ou bien de formes fréquents (traits) dans le cas d'idéogrammes ou de pictogrammes. Cette première étape consiste donc à extraire automatiquement les invariants à partir d'une collection de documents, afin que l'utilisateur puisse les utiliser dans l'étape 2. Pour extraire les invariants, les *strokes* sont extraits à partir de la collection de documents en utilisant la méthode d'extraction de *strokes* primaires par détection de zones ambiguës et le regroupement de *strokes* primaires. Puis, nous calculons les caractéristiques de ces *strokes* pour faire un *clustering*. Les prototypes des *clusters* sont les invariants. Afin de raffiner ces invariants afin de les rapprocher des souhaits/besoins des utilisateurs pour la

phase ultérieure de composition de requête, nous faisons appel à ce dernier. Celui-ci peut corriger (respectivement, explicitement et implicitement) les phases de *clustering* (pour fusionner ou découper des *clusters*) et d'extraction de *strokes* (pour découper ou regrouper des *strokes* voisins dans le document). La Figure 2.2 montre la procédure globale de l'étape « extraction d'invariants ».

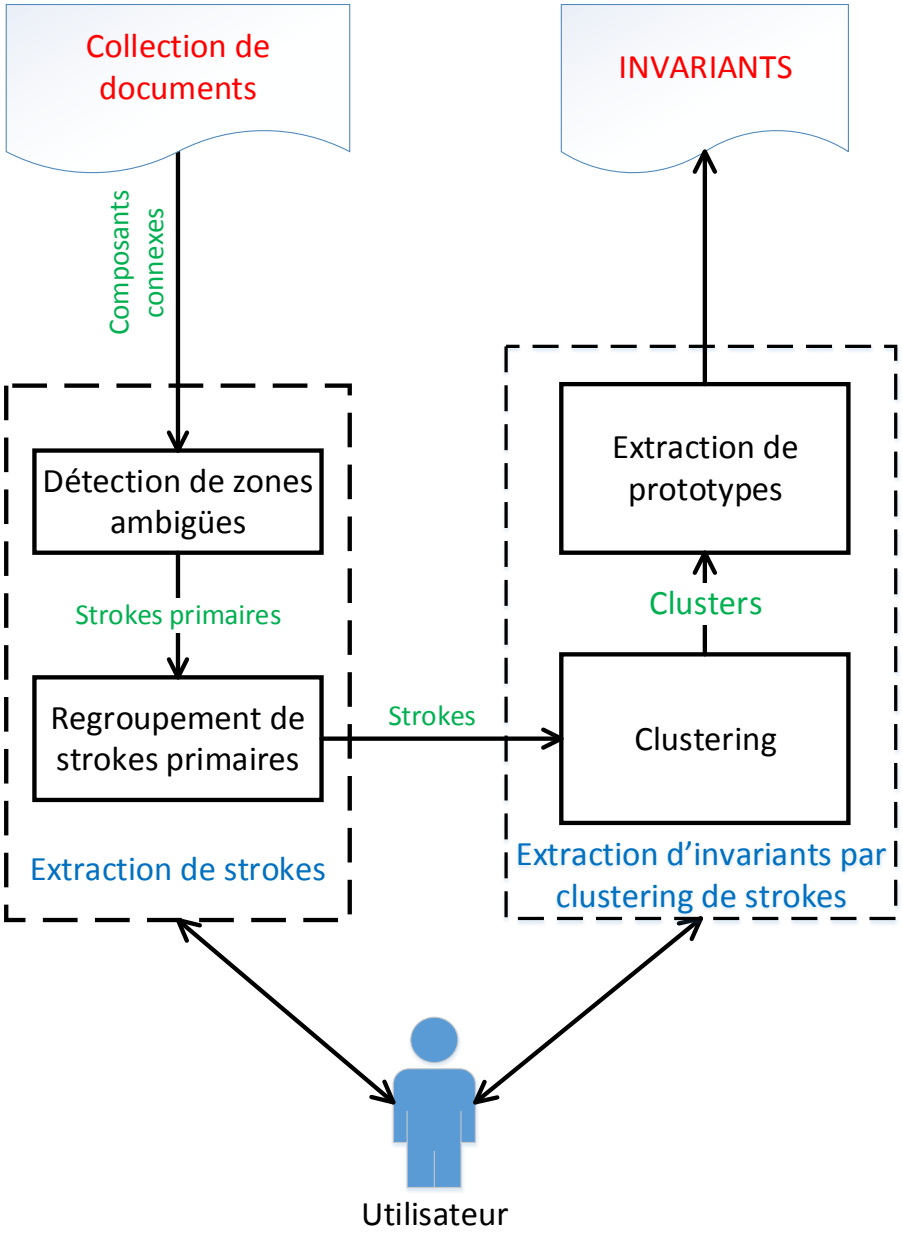


Figure 2.2 : Procédure globale de l'étape « extraction d'invariants »

- ***Étape 2 : Composition interactive de requête***

Cette étape consiste à créer le mot à rechercher par composition interactive par l'utilisateur. L'idée est de proposer à l'utilisateur une interface intuitive qui lui permette de composer sa requête quand il n'a pas trouvé une occurrence du mot à

rechercher dans le document (et donc que le word spotting traditionnel ne peut être mis en œuvre). La composition de requête se fait à partir de l'ensemble des invariants extraits dans la première étape.

- ***Étape 3 : Recherche***

En utilisant les invariants extraits (obtenu dans l'étape 1), nous définissons une signature qui nous permet de représenter les images de mots dans la collection de documents et la requête (obtenue dans l'étape 2). La similarité entre la requête et chacune des images de mots dans la collection de document est donc calculée. Les images de mots dont la similarité avec la requête est maximale sont donc retournées comme résultat de la recherche. La Figure 2.3 montre la procédure de l'étape de recherche :

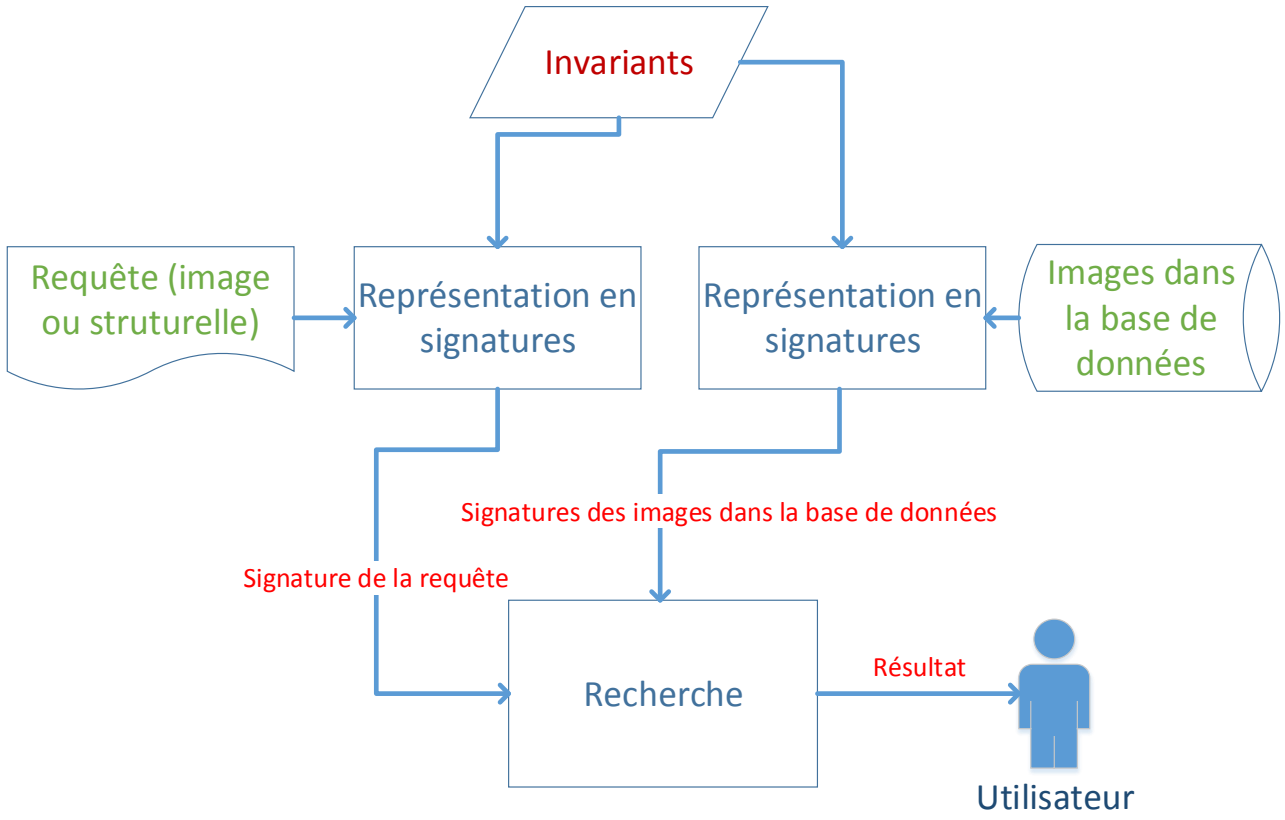


Figure 2.3 : La procédure de l'étape de recherche

L'étape 1 se fait « hors-ligne » sur la base de la collection homogène. Les étapes 2 et 3 se fait « en-ligne » avec retour rapide des mots recherchés.

Chapitre 3 : Extraction d'invariants

A secret to getting ahead is getting started

Mark Twain

3.1. Introduction

L'une de nos contributions majeures est l'introduction d'un nouveau descripteur du texte : les invariants. Les invariants sont les formes les plus fréquentes dans la collection de documents. Il peut s'agir par exemple de caractères dans le cas des langages basés sur un alphabet, ou bien de formes (traits) récurrentes dans le cas d'idéogrammes ou de pictogrammes. Nous pouvons utiliser les invariants pour décrire les images de mots lors d'une recherche. Nous pouvons également utiliser les invariants pour construire les images de requêtes pour la recherche de mots ou de portions de mots.

Nous présentons dans ce chapitre la méthode que nous proposons pour extraire les invariants. Notons que notre méthode d'extraction d'invariants est automatique et sans information *a priori* à propos du script ou du langage utilisé. Vu que les invariants seront utilisés par l'utilisateur pour construire les images de requêtes (pour la recherche de mots), nous essayons d'extraire des invariants qui fassent sens pour un humain. La Figure 3.1 montre le processus de notre méthode d'extraction d'invariants.

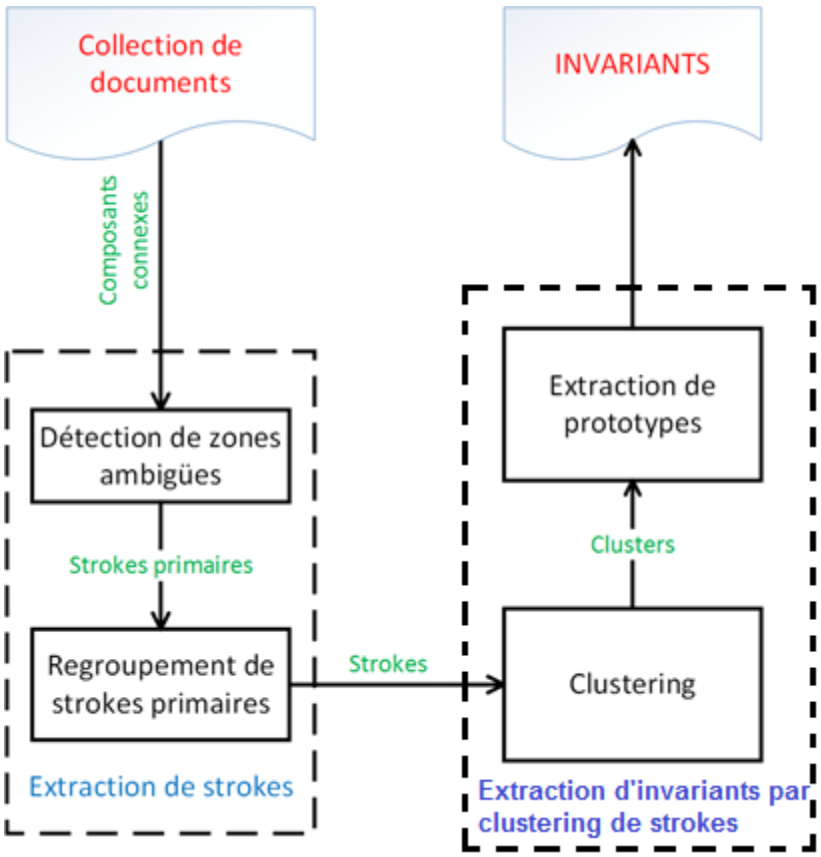


Figure 3.1 : Aperçu du processus d'extraction

La méthode d'extraction d'invariants que nous proposons repose sur 2 étapes :

- **Extraction de *strokes* :** les *strokes* sont les traits de l'écriture, récurrents dans la collection de documents. Pour extraire les invariants, nous devons extraire les *strokes* à partir de la collection de documents et puis, créer des regroupements de *strokes* similaires. Le représentant (prototype) de chaque regroupement de *strokes* pourra être considéré comme un invariant. La méthode d'extraction de *strokes* repose sur 2 étapes :
  - ***Extraction de strokes primaires basées sur la détection de zones ambiguës :*** La recherche d'invariants nécessite de trouver des zones caractéristiques dans l'écriture. Ces zones sont caractéristiques de croisement de traits, de surépaisseur, de points terminaux. Nous appelons ces zones : zones ambiguës. Pour extraire les invariants, les *strokes* primaires qui s'appuient sur les zones ambiguës doivent d'abord être extraits. Nous présentons en section 3.2 les méthodes existantes et la méthode que nous avons utilisée pour l'extraction de *strokes* primaires.
  - ***Regroupement spatial de strokes primaires :*** Après l'extraction de *strokes* primaires à partir de la collection de documents, ces *strokes*

primaires sont regroupés pour former des *strokes* de taille plus importante et que nous espérons plus en adéquation avec les usages ultérieurs de l'utilisateur (par exemple la composition de requête). Nous présentons en section 3.3 la méthode que nous avons proposée pour le regroupement spatial de *strokes* primaires.

- **Extraction d'invariants par regroupement de *strokes* :** Après l'extraction de *strokes*, ces derniers sont regroupés en *clusters* dont nous extrayons les prototypes, qui deviennent les invariants. L'annexe A offre un panorama des descripteurs de forme existants et les annexes B et C présentent respectivement les principales méthodes de *clustering* et l'approche de consensus *clustering*. La section 3.4 introduit la méthode d'extraction d'invariants par *clustering* de *strokes* que nous proposons.

Dans les sections suivantes, nous présentons les détails des étapes de notre méthode d'extraction d'invariants.

3.2. Extraction de *strokes* primaires basées sur la détection de zones ambiguës

L'extraction de *strokes* primaires passe donc naturellement par la détection de zones ambiguës. Les « *strokes* primaires » sont l'ensemble des points connexes de l'écriture entre deux extrémités ou zones ambiguës. Dans cette section, nous commençons par passer en revue les méthodes de détection de zones ambiguës existantes avant de détailler celle que nous avons retenue dans notre système.

3.2.1. Les méthodes de détection de zones ambiguës

Dans une écriture, les zones ambiguës sont celles qui appartiennent potentiellement à la conjonction de plusieurs *strokes* (voir la Figure 3.2). Par exemple, les croisements ou points de jonction de traits. L'extraction de ces zones fait l'objet de la première étape de notre système.

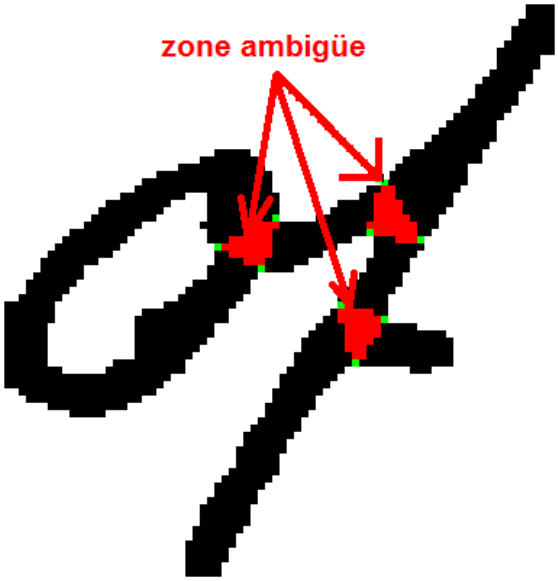


Figure 3.2 : Exemple de zones ambiguës

Nous pouvons définir une taxonomie des méthodes de la détection de zones ambiguës en 2 catégories :

- ***Les approches basées sur la squelettisation***
- ***Les approches basées sur le contour de l'image***

Nous présentons dans les parties suivantes certaines méthodes représentatives de ces deux catégories.

3.2.1.1. Les méthodes basées sur la squelettisation

La squelettisation peut être réalisée par une classe d'algorithmes visant à représenter une forme par un ensemble de courbes « fils de fer », appelé « squelette » (voir Figure 3.3). L'image de squelette d'une forme conserve les propriétés topologiques de la forme d'origine ainsi que ses propriétés géométriques. Grâce à l'image de squelette, les zones ambiguës et points d'extrémité peuvent être proposés.

Dans l'image de squelette d'une forme, il existe deux types de points particuliers : les points de croisement et les points d'extrémité. Un point de croisement est un point dans le squelette qui est la jonction d'au moins 3 branches du squelette. Un point d'extrémité est un point qui a seulement 1 point de squelette dans son voisinage. La Figure 3.3 montre des exemples de ces types de points :

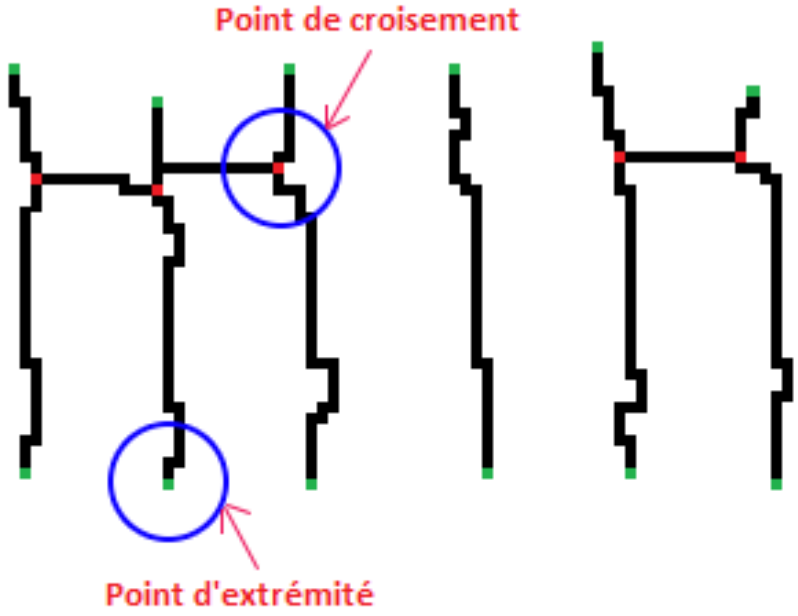


Figure 3.3 : Une image de squelette et des exemples de points de croisement et d'extrémité

Idéalement, une branche du squelette est considérée comme un trait simple (*stroke* primaire) dans l'écriture. Chaque point de croisement indique donc la présence d'une zone ambiguë. Mais, à cause de la nature intrinsèque des algorithmes de squelettisation, des fausses branches peuvent être produites (voir la Figure 3.4), souvent dans les zones ambiguës, ou dans les barbules. Ces fausses branches n'ont pas de sens au niveau de l'écriture. Donc, en réalité, il existe 3 contextes de détermination des zones ambiguës :

- *Premier cas :* Une zone ambiguë est extraite par un seul point de croisement. Ce point de croisement joint des vraies branches.
- *Deuxième cas :* une zone ambiguë n'est pas localisée par un seul, mais par un groupe de points de croisement. Les points de croisement dans ce groupe sont les connexions de fausses branches (voir la Figure 3.4). Dans ce cas-là, les fausses branches doivent être regroupées pour retrouver la zone ambiguë.
- *Troisième cas :* un point de croisement ne correspond à aucune zone ambiguë. Les points de ce type sont souvent produits dans les barbules. Dans ce cas là, il y a la présence des fausses branches. Ces fausses branches doivent être supprimées (voir la Figure 3.4)

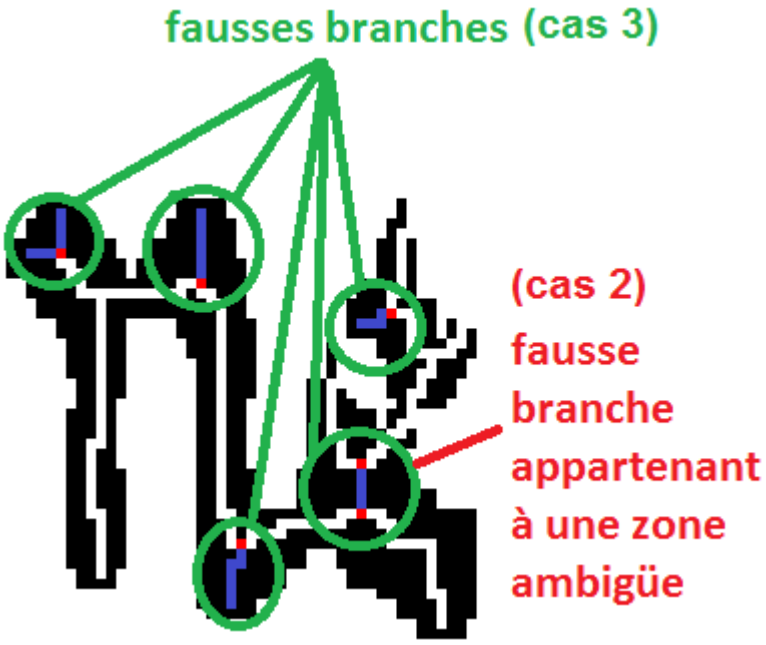


Figure 3.4 : Exemples de fausses branches

L'objectif principal de la plupart des approches de détection de zones ambiguës dans la littérature est donc de retrouver les fausses branches afin de les regrouper (pour retrouver les zones ambiguës associées à plusieurs fausses branches) ou de les enlever (lorsqu'elles appartiennent à des barbules). Des approches différentes ont été proposées dans la littérature, par exemple : l'approche « critère du cercle maximale » [Liu *et al.* 1999], [Liu *et al.* 1997] ou l'approche « double seuil » [Qiao & Yasuhara 2004; Su *et al.* 2009].

[Liu *et al.* 1999] proposent le « critère du cercle maximal » pour retrouver les fausses branches produites par l'algorithme de squelettisation. Supposons que 2 points particuliers (points de croisement ou points d'extrémité) P_1 et P_2 sont joints à une même branche $B_{P_1P_2}$. Pour le point P_1, les auteurs extraient le cercle de rayon R_1 maximal et centré sur P_1 contenant seulement des pixels d'avant-plan. En appliquant la même méthode pour le point P_2, les auteurs extraient le cercle de rayon R_2 maximal et centré sur P_2 (voir la Figure 3.5). Selon le critère proposé par les auteurs, si la distance entre P_1 et P_2 est inférieure ou égale à la somme de R_1 et R_2 ($distance(P_1, P_2) \leq R_1 + R_2$), alors la branche $B_{P_1P_2}$ est une fausse branche.

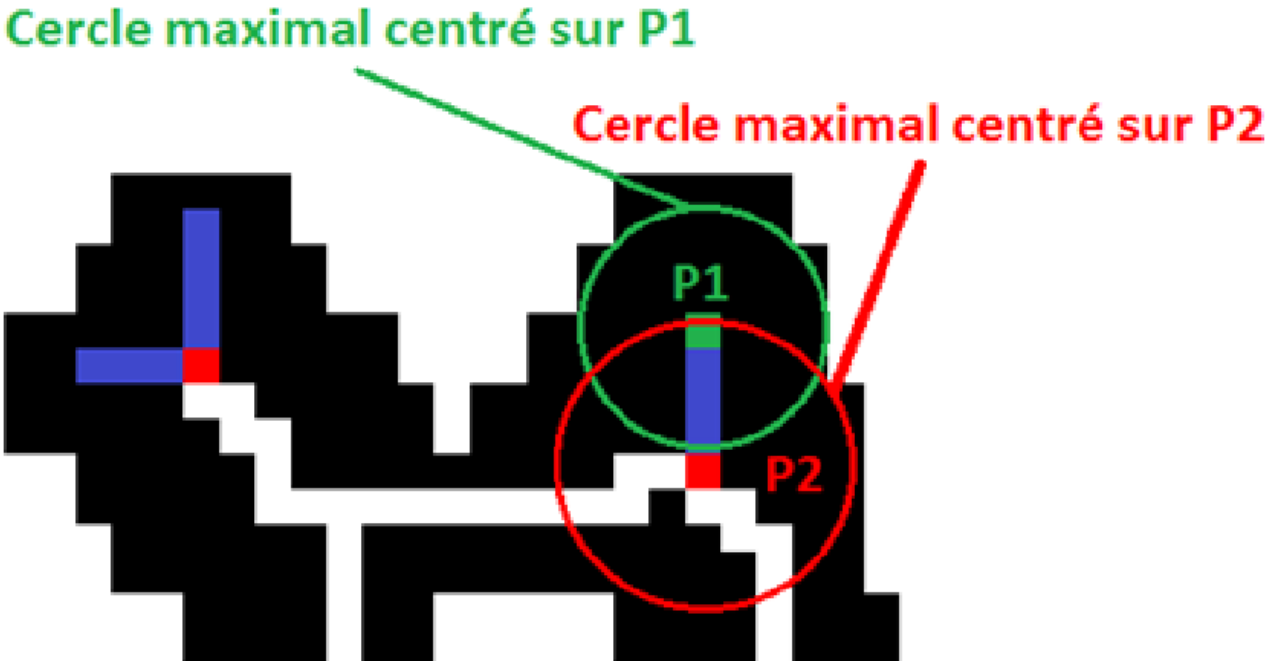


Figure 3.5 : Extraction des cercles maximaux selon [Liu *et al.* 1999]

Pour retrouver les fausses branches, [Kato & Yasuhara 2000] mettent en œuvre un seuil l_{th}, proportionnel à l'épaisseur de l'écriture moyenne estimée w : $l_{th} = C.w$ (C est une constante prédéfinie). Si la distance entre 2 points particuliers d'une branche est inférieure à l_{th}, alors cette branche est considérée comme une fausse branche. L'épaisseur de l'écriture moyenne est estimée en calculant la distance moyenne des distances de chaque point du squelette au pixel du contour le plus proche (voir la Figure 3.6),

Les méthodes de [Liu *et al.* 1999], [Kato & Yasuhara 2000] tendent souvent à retourner des faux positifs (cas où une vraie branche de faible longueur est détectée comme une fausse branche). Pour contourner ce problème, [Qiao & Yasuhara 2004] utilisent la technique dite du « double seuil » pour retrouver les fausses branches. Cette méthode est une amélioration de la méthode de [Kato & Yasuhara 2000] qui utilisent 2 seuils sur la longueur de la branche. Plus précisément, les auteurs introduisent deux seuils prédéfinis : l_{th1} et l_{th2} ($l_{th1} > l_{th2}$). Les deux seuils : l_{th1} et l_{th2} sont fixés par les auteurs ($l_{th1} = 4w, l_{th2} = 1{,}5w$, où w est l'épaisseur de l'écriture moyenne estimée). Supposons que 2 points spéciaux P_1 et P_2 sont connectés à une branche $B_{P_1P_2}$. Si la longueur de la branche $B_{P_1P_2}$ est supérieure à l_{th1} alors la branche $B_{P_1P_2}$ est une branche réelle. Si la longueur de la branche $B_{P_1P_2}$ est inférieure à l_{th2}, alors, la branche $B_{P_1P_2}$ est une fausse branche. Si la longueur de la branche $B_{P_1P_2}$ est comprise l_{th1} et l_{th2}, alors, pour chaque point p_i dans la branche, les auteurs calculent la distance minimale entre le point p_i et les points dans le contour de la forme : $distance(p_i, C)$ (voir

Figure 3.6). Si $\sum_{p_i \in B_{P_1P_2}} distance(p_i, C)/|B_{P_1P_2}| < 0{,}65\, w$ et $\max_{p_i \in B_{P_1P_2}} distance(p_i, C) < w$ alors la branche $B_{P_1P_2}$ est une vraie branche, sinon, $B_{P_1P_2}$ est une fausse branche.

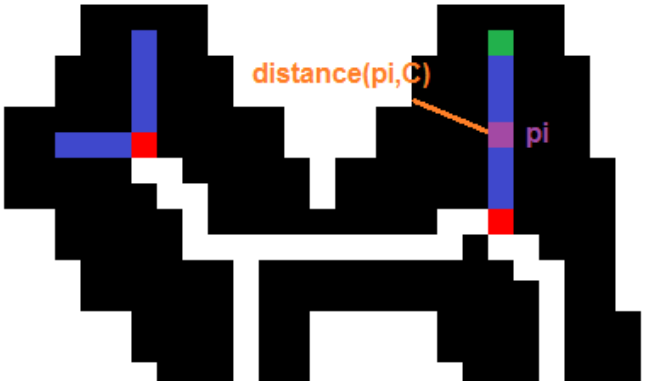


Figure 3.6 : Distance minimale d'un point au contour de l'image

Après la détection de fausses branches, les auteurs de [Liu *et al.* 1999], [Qiao & Yasuhara 2004], [Kato & Yasuhara 2000] utilisent la même méthode pour détecter les zones ambiguës. Chaque point de croisement qui joint seulement des vraies branches indique la présence d'une zone ambiguë. Puis, ils regroupent les fausses branches connectées entre elles, avec les points de croisement associés dans un même groupe. Chaque groupe indique la présence d'une zone ambiguë. Enfin, les fausses branches qui ne sont pas connectées entre elles sont enlevées, car considérés comme appartenant à des barbules.

[Su *et al.* 2009] utilisent la même technique du « double seuil » que la méthode de [Qiao & Yasuhara 2004] pour retrouver les fausses branches. Mais [Su *et al.* 2009] n'extraient que les fausses branches qui ne correspondent à aucune zone ambiguë pour les enlever. Afin de retrouver les zones ambiguës, les auteurs de [Su *et al.* 2009] utilisent l'algorithme demi-squelettisation [Kim & Kim 2003] pour regrouper les points de croisement. Chaque groupe de points de croisement indique la présence d'une zone ambiguë. La deuxième différence entre la méthode [Qiao & Yasuhara 2004] et la méthode [Su *et al.* 2009] est l'amélioration de la technique « double seuil ». Plus précisément, un problème critique concernant la méthode [Qiao & Yasuhara 2004], c'est que si une fausse branche assez longue ($l_{th2} > B_{P_1P_2} > l_{th1}$) produite à cause d'un changement important dans l'orientation de l'écriture, alors cette fausse branche n'est pas détectée. Dans la méthode de [Su *et al.* 2009], ce problème est résolu dans l'étape de détection de zone ambiguë.

Conclusion :

Nous avons présenté dans cette partie quelques-unes des approches basées sur la squelettisation pour l'extraction de zones ambiguës. En général, les approches basées sur la squelettisation sont rapides en terme de temps de calcul. Néanmoins, ces approches dépendent de l'uniformité de l'épaisseur de l'écriture et de l'absence de bruits dans le trait. Avec des documents dans lesquels l'écriture a une variabilité élevée dans sa représentation des traits (épaisseur), la performance des méthodes basées sur la squelettisation est donc peu élevée. Pour contourner notamment ce problème, les approches basées sur le contour de l'image ont été proposées. Nous présentons ces approches dans la section suivante.

3.2.1.2. Les méthodes basées sur le contour de l'image

Dans ces approches, les points du contour de l'image et leurs caractéristiques sont utilisées pour la détection des zones ambiguës. Parmi ces points de contour, il y a des « points dominants » qui correspondent à des changements brusques de la courbure du contour. [Lee & Wu 1998], [Plamondon & Privitera 1999] ont démontré que les zones ambiguës peuvent être localisées en utilisant les points dominants, comme détaillé dans la suite.

[Lee & Wu 1998] extraient les points dominants en utilisant l'algorithme de détection d'angle de [Teh & Chin 1989]. Les points dominants divisent le contour en segments. Pour chaque segment du contour, les auteurs trouvent le segment opposé (voir Figure 3.7) en utilisant l'estimation de la tangente de chaque point dans le segment. La méthode utilisée pour trouver le segment opposé est détaillée dans [Lee & Wu 1998]. Les auteurs définissent les régions de *strokes* comme délimitées par une paire de segments opposés (voir Figure 3.7). Les régions restantes sont les zones ambiguës (voir Figure 3.7).

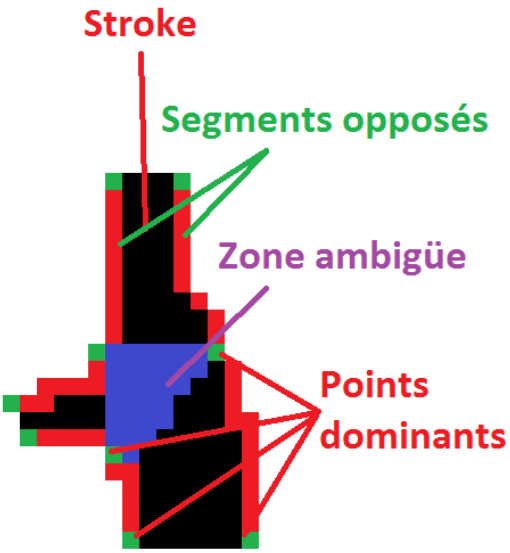


Figure 3.7 : Points dominants, segments opposés et zones ambiguës

L'inconvénient de la méthode de [Lee & Wu 1998] réside dans sa dépendance forte vis-à-vis de l'extraction des points dominants qui est peut-être peu fiable en présence de bruits et de distorsions. Pour diminuer cette dépendance et donc pour rendre le système plus stable, [Plamondon & Privitera 1999] regroupent les points dominants. Chaque groupe de points dominants indique une zone ambiguë. Plus précisément, tout comme [Lee & Wu 1998], les auteurs extraient également les points dominants en utilisant l'algorithme de [Teh & Chin 1989]. Après l'extraction des points dominants, les zones ambiguës sont détectées. Mais, cette fois-ci, la méthode de détection de zones ambiguës est une méthode itérative qui détecte d'abord la zone d'interférence (la zone de croisement de *strokes*) en déplaçant une fenêtre glissante de gauche à droite. La fenêtre circonscrit une zone de croisement de *strokes* si et seulement si les *strokes* croisent la bordure de la fenêtre plus de 2 fois et s'il existe seulement 1 composante connexe dans la fenêtre. La Figure 3.8 illustre ce mécanisme sur un exemple. Dans la Figure 3.8 (a), les *strokes* croisent la bordure de la fenêtre 4 fois, et il existe seulement 1 composante connexe dans la fenêtre. Dans la Figure 3.8 (b), la fenêtre ne circonscrit pas une zone d'interférence même si les *strokes* croisent la bordure de la fenêtre 4 fois, parce qu'il existe 2 composantes connexes dans la fenêtre.

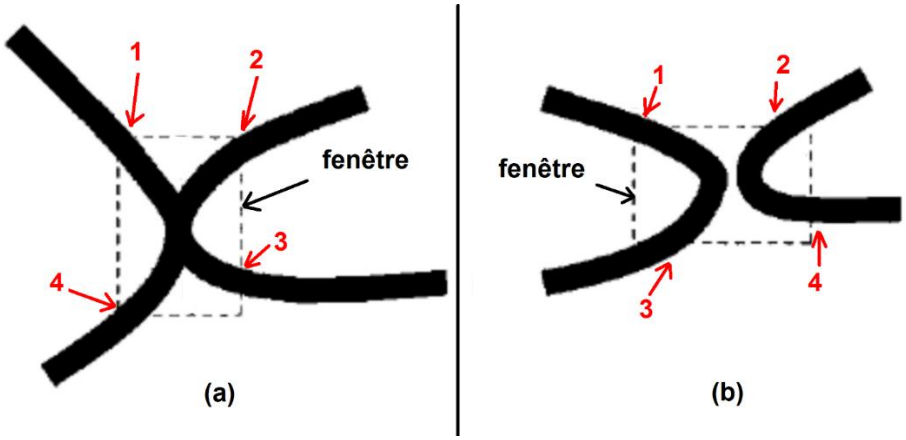


Figure 3.8 : (a) la fenêtre circonscrit une zone d'interférence. (b) la fenêtre ne circonscrit pas une zone d'interférence

Initialement, la taille de la fenêtre est fixée heuristiquement pour tous les documents de la collection de manière à ce que la fenêtre puisse circonscrire les zones d'interférence. Cette taille n'est pas modifiée jusqu'à ce qu'une zone d'interférence soit détectée. Dès qu'une zone d'interférence est détectée, [Plamondon & Privitera 1999] modifient la taille de la fenêtre de manière à lui attribuer la taille minimale qui couvre tous les points dominants dans cette zone d'interférence. La zone ambiguë est la zone de l'écriture qui est couverte par la fenêtre minimale trouvée (voir la Figure 3.9). Après la détection d'une zone ambiguë, les auteurs réinitialisent la taille de la fenêtre et continuent à déplacer la fenêtre pour trouver les autres zones ambiguës.

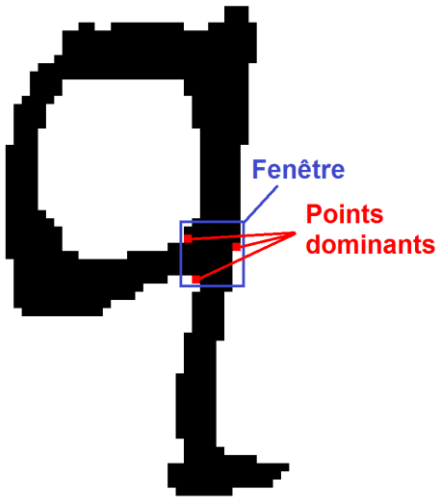


Figure 3.9 : Méthode de [Plamondon & Privitera 1999] **pour détecter les zones ambiguës**

[Homer 2000] considère un *stroke* est comme une famille des cercles $C^* = C_t(O(t), r(t))$ où $O(t)$ est le centre et $r(t)$ est le rayon du cercle C_t. (Voir Figure 3.10).

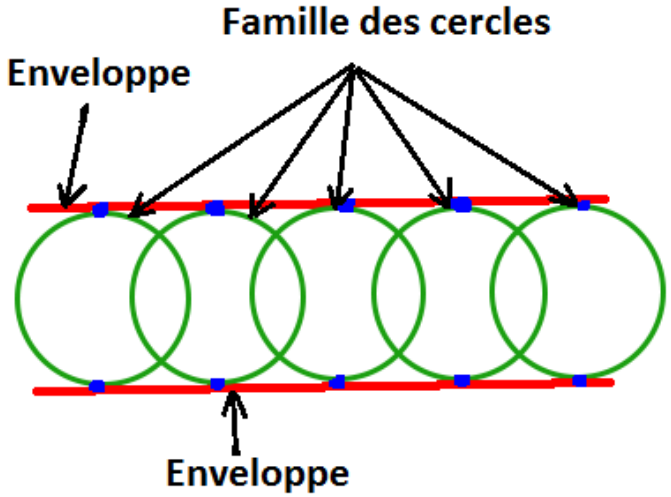


Figure 3.10 : Une famille des cercles représentant d'un *stroke* et ses enveloppes

Les auteurs définissent une courbe Γ comme l'enveloppe de C^*. Une courbe Γ est l'enveloppe d'une famille des cercles $C^* = C_t(O(t), r(t))$ si et seulement si Γ est tangente en chacun de ses points à un cercle C_t de la famille C^* et toute cercle dans la famille C^* est tangente en au moins un point à Γ (voir la Figure 3.10). Les auteurs définissent deux courbes Γ_1 et Γ_2 comme deux enveloppes opposées de la famille des cercles C^* (voir la Figure 3.10). Dans les zones des *strokes* explicites (les zones qui ne sont pas ambiguës), les points des enveloppes Γ_1, Γ_2 sont contenus dans le contour. Dans les zones ambiguës où l'information du *stroke* est incertaine, les points des enveloppes Γ_1, Γ_2 ne sont pas contenus dans le contour. Ces enveloppes sont cachées dans la région des zones ambiguës (voir la Figure 3.11).

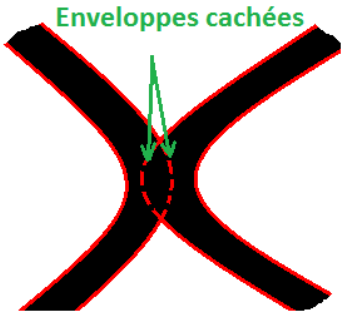


Figure 3.11 : Enveloppes cachées

Pour retrouver les zones ambiguës, [Homer 2000] essaient de trouver les endroits où les points des enveloppes ne sont pas contenus dans le contour. Plus précisément, pour chaque point p de l'enveloppe Γ_1 contenu dans le contour, les auteurs trouvent le point opposé p' dans le contour. Si le point p' n'est pas contenu dans l'enveloppe Γ_2, Γ_2 est cachée dans une zone ambiguë. Autrement dit, une zone ambiguë est localisée. Pour retrouver le point opposé p', les auteurs utilisent une estimation de tangente au point p. Pour vérifier si p' est contenu dans l'enveloppe Γ_2, les auteurs appliquent une estimation de courbure et d. Pour plus de détail sur la méthode d'extraction de zones ambiguë, nous renvoyons le lecteur à [Homer 2000].

Un inconvénient commun des méthodes basées sur le contour (comme [Lee & Wu 1998], [Plamondon & Privitera 1999], [Homer 2000]) est qu'elles relient sur l'estimation de tangente et de courbure ce qui ne sont pas fiables sur la présence de bruits dans le cas des documents dégradés.

Conclusion :

Nous avons présenté dans cette partie quelques-unes des méthodes d'extraction des zones ambiguës basées sur l'analyse de contour. Les méthodes de ce type ne sont pas sensibles à l'estimation de l'épaisseur de l'écriture. Leur performance ne dépend donc pas des variations de l'écriture. Mais, en général, leur complexité est plus élevée que celle des méthodes basées sur la squelettisation. De plus, les méthodes basées sur le contour ne sont pas fiables dans le cas de documents dégradés.

3.2.2. Sélection de la méthode la mieux adaptée à notre contexte

Pour extraire les *strokes* primaires, nous cherchons à sélectionner, parmi les méthodes de la littérature, la méthode de détection de zone ambiguë la plus adaptée à notre contexte applicatif. Pour cela, nous nous sommes basé sur une étude expérimentale présentée ci-dessous.

3.2.2.1. Comparaison expérimentale

Nous avons comparé expérimentalement, sur diverses collections de documents de caractéristiques différents, 2 méthodes de natures différentes :

- La méthode de [Su *et al.* 2009], basée sur la squelettisation.
- La méthode de [Plamondon & Privitera 1999], basée sur le contour.

Nous choisissons ces 2 méthodes parce qu'elles sont représentatives des différents types d'approches de la littérature. La méthode de [Su *et al.* 2009] est représentative des méthodes basées sur la squelettisation. La méthode de [Plamondon & Privitera 1999] est représentative des méthodes basées sur le contour et utilisant une estimation de la courbure.

Dans nos expérimentations, nous utilisons 3 collections de document :

- Un extrait de la base « Saint Gall » [1] qui contient 2 pages des documents anciens manuscrits (mono-scripteur) en latin. Elles sont écrites au 9ième siècle et contenant 2051 composantes connexes.
- Un extrait de la base « Chinois » qui contient 2 pages extraites du livre manuscrit chinois : « Bai shi wen ji » [2], imprimé en 1618, et contenant 934 composantes connexes.
- Un extrait de la base « Indien » qui contient 2 pages extraits du livre imprimé indien « Bhavishya Puranam » [3], imprimé en 1954 et contenant 1933 composantes connexes.

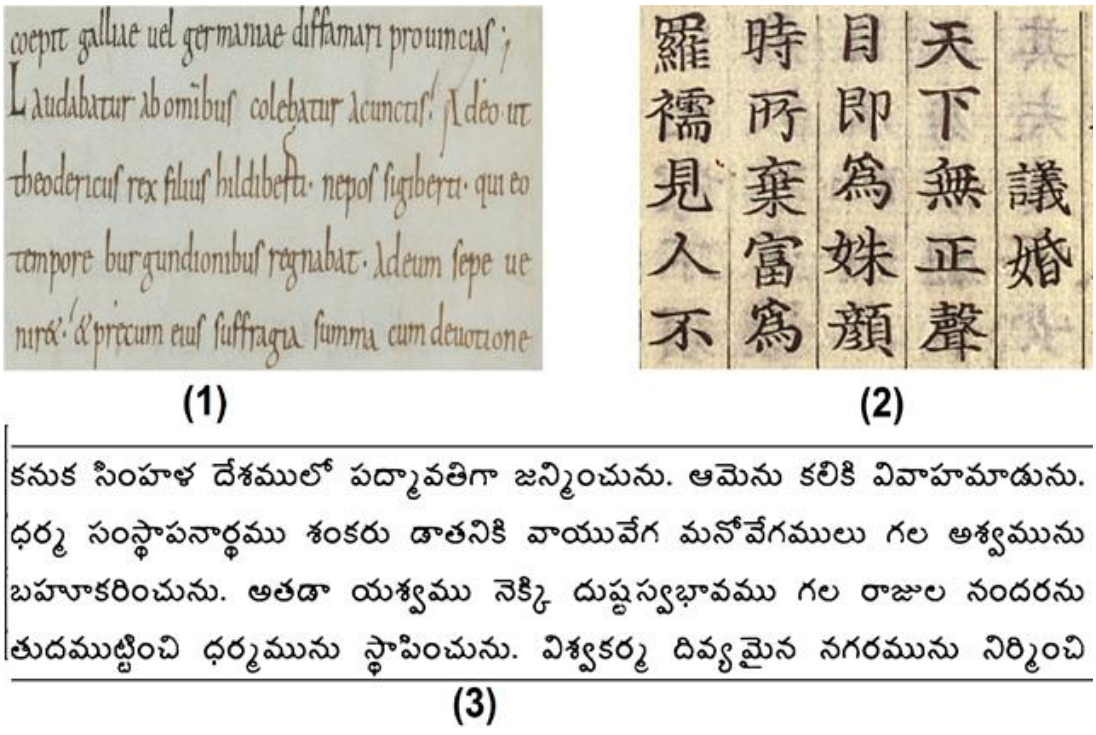


Figure 3.12 : Extraits des 3 bases de données utilisées. 1) La base « Saint Gall ». 2) La base « Chinois ». 3) La base « Indien »

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/saint-gall-database
[2] http://www.ndl.go.jp/exhibit60/e/copy1/4rekishi_2.html
[3] https://archive.org/details/bhavishyapuranam014600mbp

La vérité-terrain a été créée semi automatiquement sous la supervision de chercheurs de notre laboratoire. Elle indique la position des zones ambiguës pour chaque composante connexe. Au sens de ce que nous avons présenté au paragraphe précédent, nous évaluons les différentes méthodes avec non seulement des documents anciens (base « Saint Gall » et base « Chinois »), mais aussi avec des documents modernes (base « Indien »), pour mesurer la robustesse en fonction du niveau de dégradation du document. Nous utilisons trois scripts différents afin d'évaluer la généricité des algorithmes comparés.

Pour chaque document, nous appliquons, dans un premier temps, un prétraitement pour binariser l'image. Plus précisément, nous appliquons la méthode de [Wolf *et al.* 2002]. Puis, les composantes connexes sont extraites. Sur chaque composante connexe, les zones ambiguës sont localisées en utilisant les deux méthodes comparées.

Nous comparons les résultats de ces méthodes avec la vérité-terrain. La pertinence des zones détectées est déterminée par l'algorithme suivant :

- Chaque zone détectée est mise en correspondance avec une zone de référence dans la vérité-terrain pour que le nombre total des pixels de l'erreur (les pixels appartient à la zone détectée mais n'appartient pas à la zone de référence et vice-versa) soit minimal.
- Supposons qu'une zone détectée Z_d est mise en correspondance avec la zone de référence Z_r ; $P(Z_d)$ est l'ensemble des pixels dans Z_d ; $P(Z_r)$ est l'ensemble des pixels dans Z_r.
- La zone Z_d est considérée comme mal détectée si aucune zone de référence n'est mise en correspondance avec elle, ou si $|P(Z_d) \backslash P(Z_r)| > |P(Z_r)|$.
- La zone Z_d est considérée comme non détectée si $|P(Z_d) \cap P(Z_r)| < 0{,}5\ |P(Z_r)|$.

Pour chaque base et pour chaque méthode, nous calculons le nombre de zones ambiguës bien détectées, le nombre de zones ambiguës mal détectées et le nombre de zones ambiguës non détectées. La précision et le rappel sont ensuite calculés comme suite :

$$Précision = \frac{|\{zones\ bien\ détectées\}|}{|\{zones\ bien\ détectées\}| + |\{zones\ mal\ détectées\}|}$$

$$Rappel = \frac{|\{zones\ bien\ détectées\}|}{|\{zones\ bien\ détectées\}| + |\{zones\ non\ détectées\}|}$$

En modifiant les paramètres de chaque méthode (appliquée sur une base de données), nous obtenons différentes valeurs de précision et de rappel. Nous pouvons donc faire la courbe de précision – rappel pour chaque méthode et pour chaque base.

La Figure 3.13 montre les courbes de précision – rappel de la méthode de [Su *et al.* 2009] et la méthode de [Plamondon & Privitera 1999], appliquées sur la base « Saint Gall ».

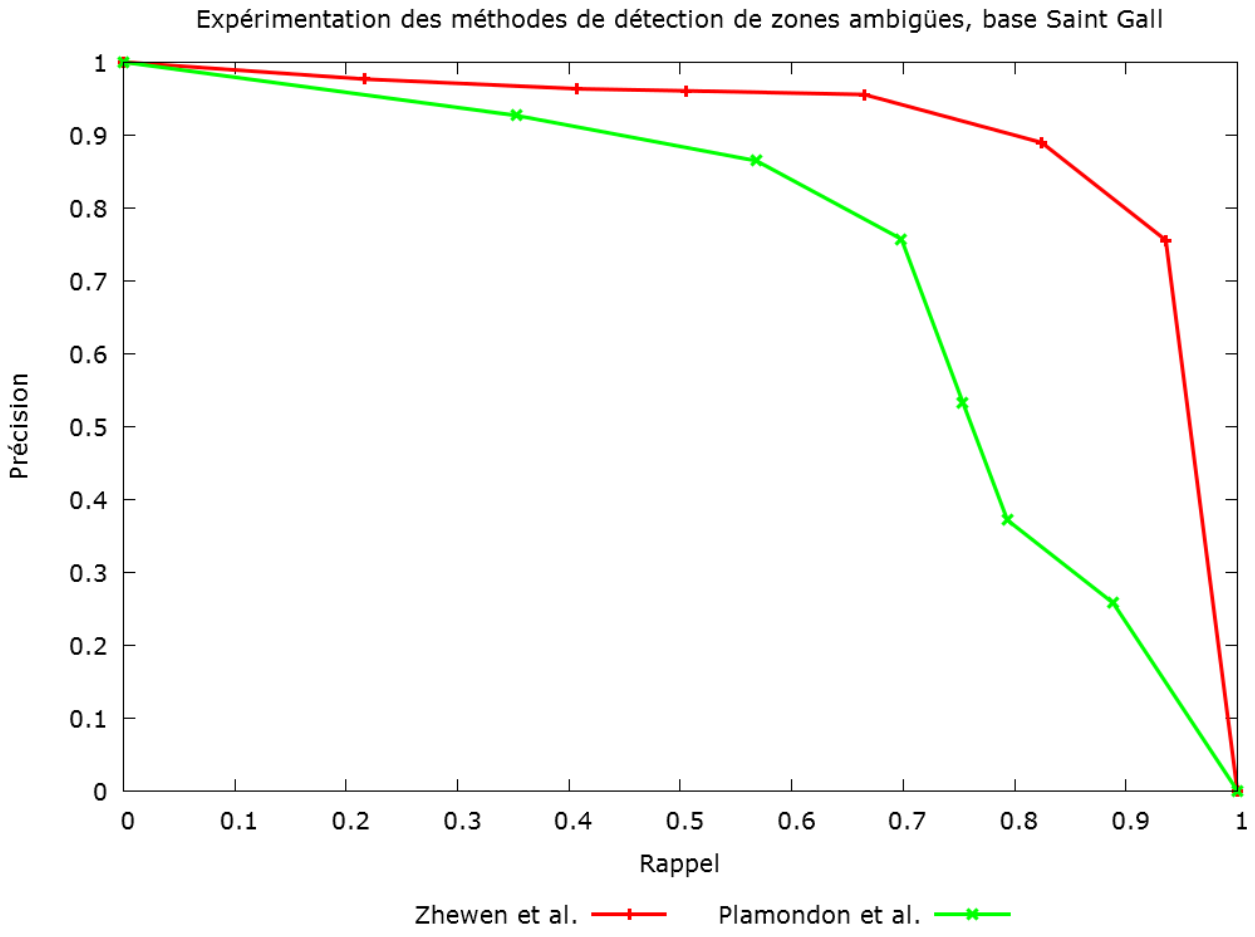


Figure 3.13 : Expérimentation des méthodes de détection de zones ambiguës, base de données : « Saint Gall »

Sur cette base très dégradée, nous pouvons constater que, la méthode de [Plamondon & Privitera 1999] donne de moins bons résultats que celle de [Su *et al.* 2009].

La Figure 3.14 montre les courbes de précision – rappel de la méthode de [Su *et al.* 2009] et la méthode de [Plamondon & Privitera 1999], appliquées sur la base « Chinoise ».

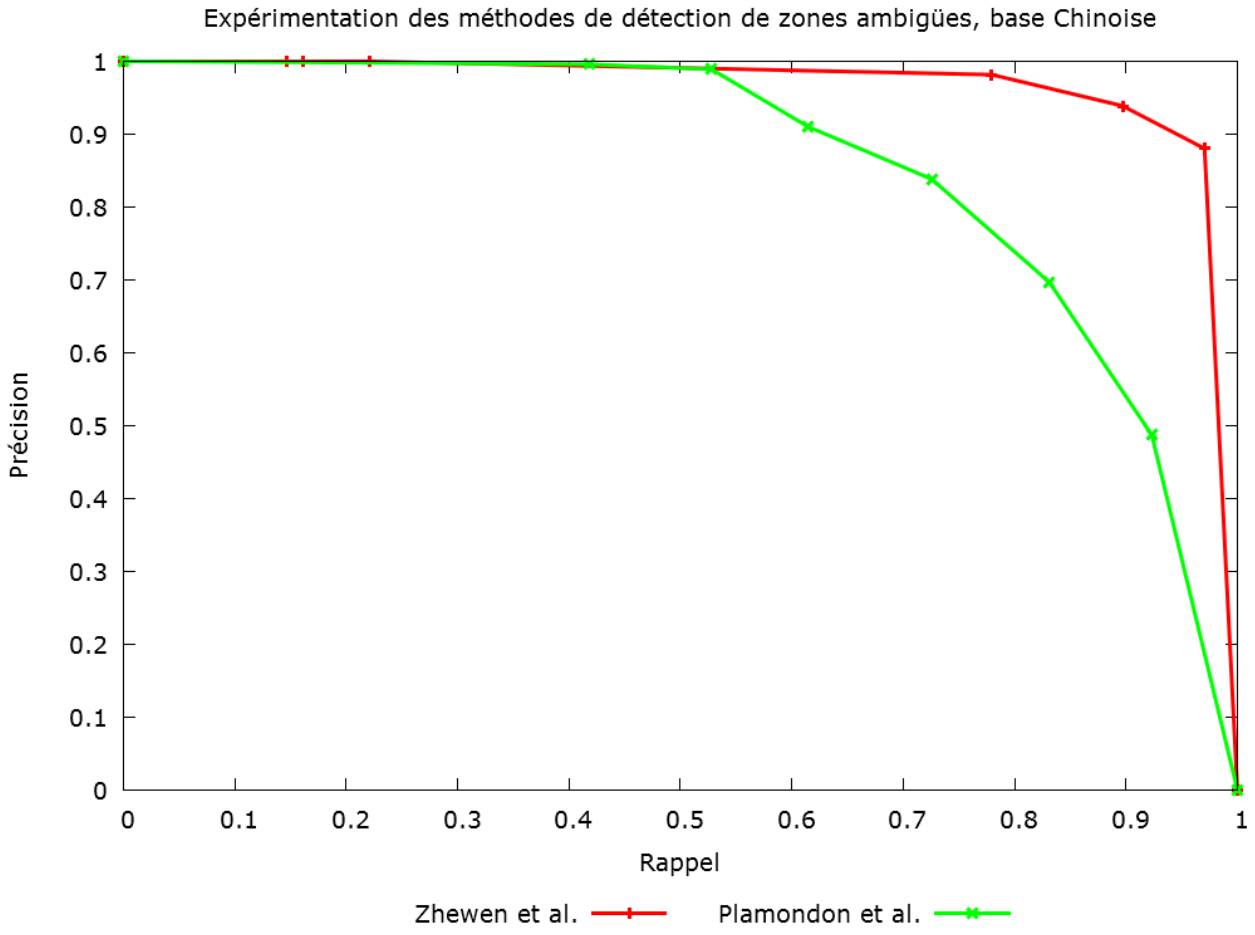


Figure 3.14 : Expérimentation des méthodes de détection de zones ambiguës, base de données : « Chinois »

Avec la base « Chinois » qui est moins dégradée que la base « Saint Gall », les deux méthodes donnent de mieux résultats qu'avec la base « Saint Gall ». Nous pouvons aussi constater que la méthode de [Plamondon & Privitera 1999] donne de moins bons résultats que celle [Su *et al.* 2009].

La Figure 3.15 montre les courbes de précision – rappel de la méthode de [Su *et al.* 2009] et la méthode de [Plamondon & Privitera 1999], appliquées sur la base « Indien ».

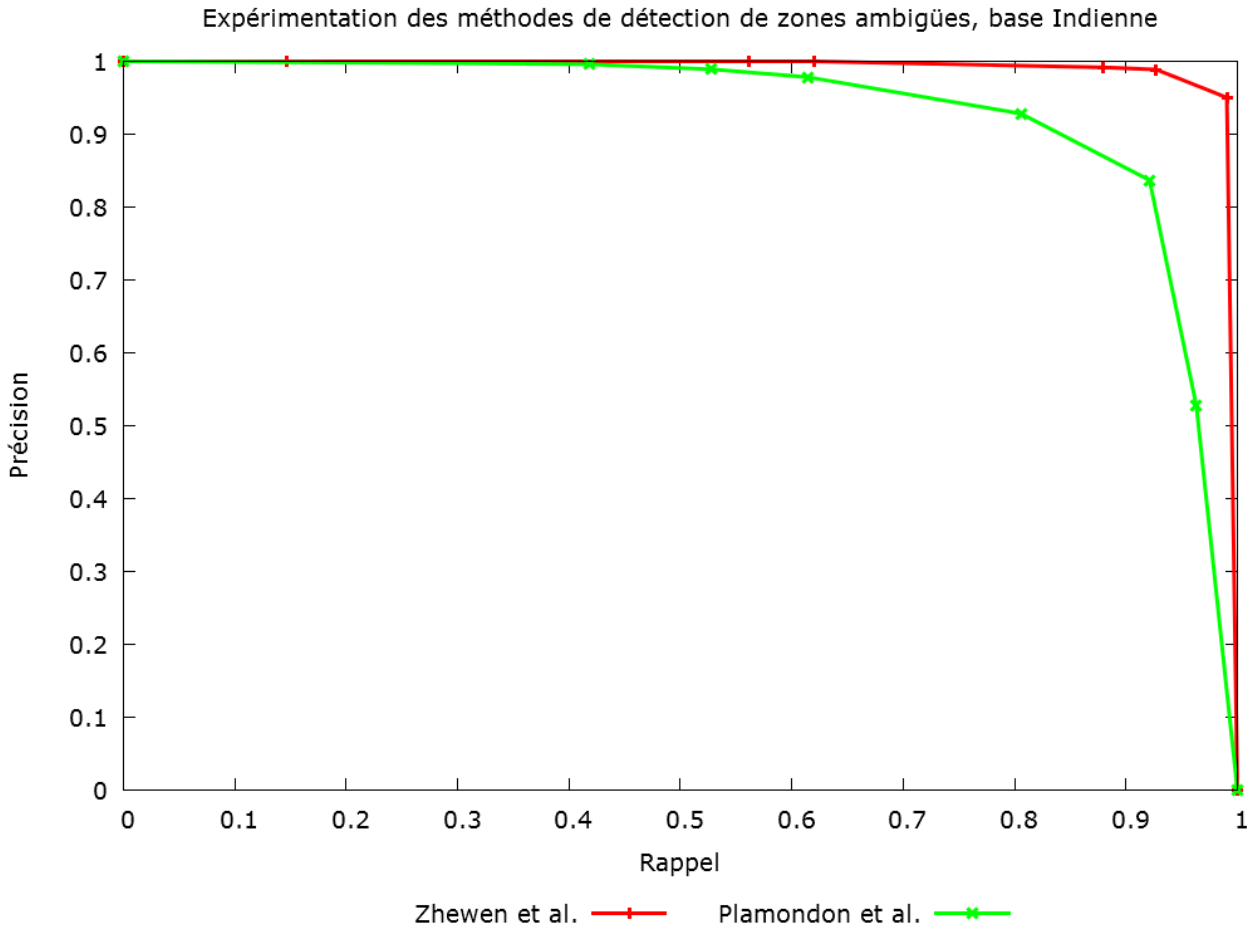


Figure 3.15 : Expérimentation des méthodes de détection de zones ambiguës, base de données : « Indien »

Avec la base « Indien » qui est la base la plus homogène et la moins dégradée, les deux méthodes donnent les résultats les meilleurs en comparaison avec les deux autres bases. Nous pouvons constater que la méthode de [Plamondon & Privitera 1999] donne de moins bons résultats que celle [Su *et al.* 2009].

3.2.2.2. Discussion

La méthode de [Plamondon & Privitera 1999] est liée à l'estimation de la courbure, ce qui la rend peu robuste en présence de documents dégradés. C'est en particulier le cas pour la base Saint Gall et pour la base « Chinois » (qui sont les bases les plus dégradées), tandis qu'elle donne le meilleur résultat sur la base Indien qui est la plus moderne et la moins dégradée.

La méthode de [Su *et al.* 2009] est efficace avec les 3 bases de scripts différents. La précision est généralement meilleure que la méthode de [Plamondon & Privitera 1999]. Ces bons résultats peuvent être expliqués par le fait que les contenus de

chacune de ces 3 bases sont assez homogènes. En effet, dans un même document manuscrit et particulièrement lorsque celui-ci est ancien, les auteurs tentent à ne pas modifier l'épaisseur de l'écriture. Dans le cas de documents imprimés, l'épaisseur de l'écriture reste presque constante. De plus, l'impact des bruits est réduit en appliquant une technique de regroupement de fausses branches. C'est un avantage de cette méthode sur la méthode de [Plamondon & Privitera 1999], qui est moins robuste en présence des documents dégradés. Sur la base de ces expérimentations, nous pouvons conclure que la méthode de [Su *et al.* 2009] est la plus adaptée à notre contexte où nous cherchons à décrire des documents anciens assez homogènes (mono-scripteur ou même imprimeur) mais potentiellement dégradés. Nous utilisons donc cette méthode pour la détection des zones ambiguës. Nous détaillons dans la section suivante cette méthode, qui a été présentée brièvement en section 3.2.1.1.

3.2.2.3. Présentation de la méthode sélectionnée

Notre étude expérimentale nous a permis de conclure que la méthode de [Su *et al.* 2009] est la plus adaptée à notre contexte. Nous utilisons donc cette méthode pour la détection des zones ambiguës. L'objectif de cette méthode est de regrouper les points de croisement du squelette qui indiquent la présence des zones ambiguës.

Dans l'image de squelette d'une forme, il y a deux types de points particuliers : les points de croisement et les points d'extrémité. Un point de croisement est un point dans le squelette qui est la jonction d'au moins 3 branches du squelette (voir la Figure 3.3). Un point d'extrémité est un point qui a seulement 1 point de squelette dans son voisinage (voir la Figure 3.3). Les définitions de l'ensemble des points de croisement du squelette S_f et de l'ensemble des points d'extrémité S_e sont les suivantes :

$$S_f = \{p | (N_c(p) >= 3 \; ou \; N_b(p) >= 4\}$$

$$S_e = \{p | N_b(p) = 1\}$$

Où p est un point dans l'image de squelette, $N_c(p)$ est le nombre de branches autour de p, $N_b(p)$ est le nombre de points dans le 8-voisinage de p.

Les points de croisement du squelette sont appelés « points de croisement candidats » (PCC). Généralement, un PCC ou un groupe de PCCs indique la présence d'une zone ambiguë. Il y a aussi des PCC qui sont produits à cause de ruptures brusques dans la direction de l'écriture ou la présence de bruits dans le document et qui ne présentent aucune zone ambiguë. Ces points se sont appelés « faux points de croisement » (FPC). Pour détecter les zones ambiguës, les points

FPC doivent être supprimés de la liste de PCC. [Su *et al.* 2009] utilisent les règles suivantes pour localiser les points FPC :

- *Première règle :* Dans le squelette, si une branche entre un point PCC et un point d'extrémité est trop courte, alors ce point PCC est produit à cause de la présence de barbules ou de bruits dans le document. Ce point est donc un point FPC.
- *Deuxième règle :* Dans le squelette, si les points dans une branche entre un point PCC et un point d'extrémité sont assez loin du contour de l'écriture (voir Figure 3.16), alors ce point PCC est produit à cause des ruptures brusques dans l'orientation de l'écriture. Ce point est donc un point FPC.

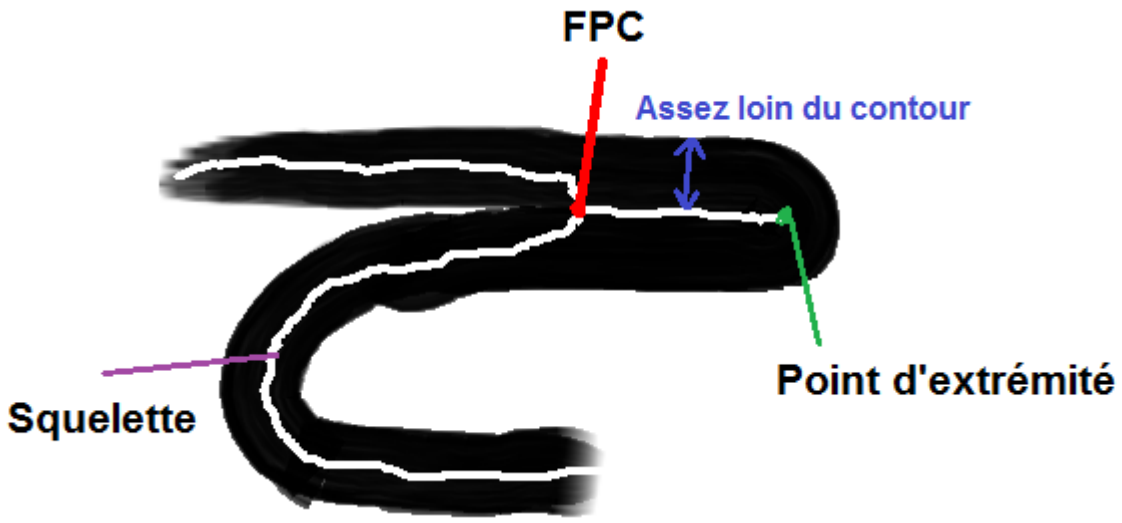


Figure 3.16 : Exemple d'un FPC

Ces deux règles sont mise en œuvre à l'aide de 3 seuils fixés en fonction de l'épaisseur moyenne de trait estimée w. Pour plus de détail sur le choix de ces seuils, nous renvoyons le lecteur à [Su *et al.* 2009].

Lorsque les points FPC sont localisés et supprimés, les points PCC restants sont regroupés pour retrouver les zones ambiguës. Supposons que S_f soit l'ensemble de tous les PCC et S_1 soit l'ensemble des FPC. Nous appelons $S_2 = S_f - S_1$. Les points dans S_2 sont regroupés par la méthode suivante afin de détecter les zones ambiguës :

- Supposons que p_0 et p_1 soient deux PCC de S_2. S'il y a une branche entre p_0 et p_1 dont la longueur est trop courte, alors p_0 et p_1 appartiennent à un même groupe. Nous réutilisons ici le même seuil que celui de la première règle pour trouver un FPC.

- Appliquer l'algorithme de demi-squelettisation [Kim & Kim 2003] sur l'image originale (voir la Figure 3.17). S_3 est l'ensemble des points de contour de l'image de demi-squelette. S_4 est l'ensemble des points de l'image de squelette. Notons $S_5 = S_2 \cap (S_4 - S_3)$ l'ensemble des PCC appartenant au squelette mais pas au contour du demi-squelette. S'il y a deux points de S_5 connectés dans l'ensemble $(S_4 - S_3)$, alors ces deux points appartiennent à un même groupe et donc à la même zone ambiguë.

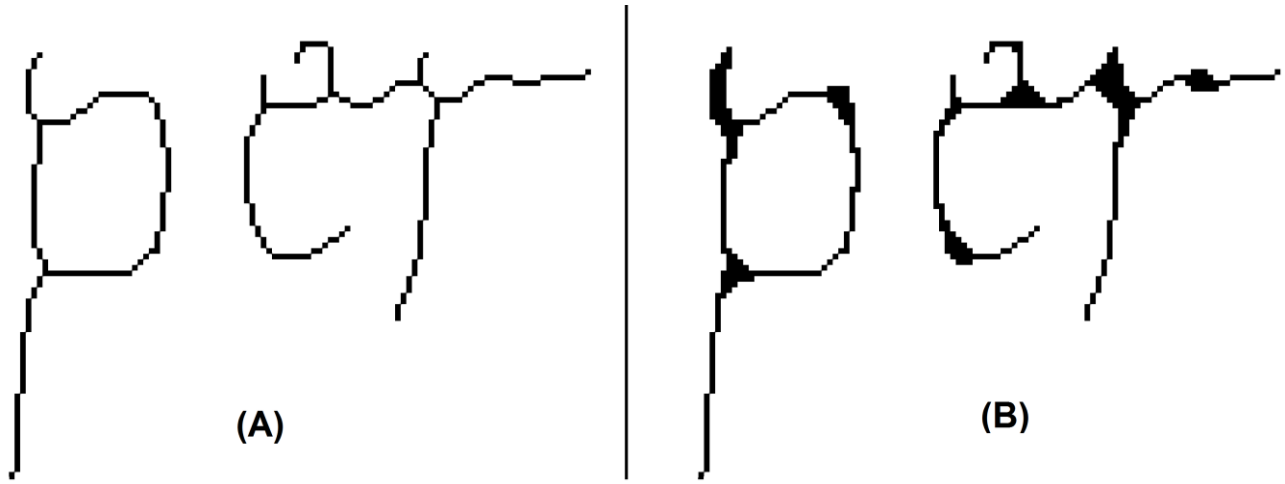


Figure 3.17 : Exemple d'une image de squelette (A) et d'une image de demi-squelette (B)

La forme de la zone ambiguë correspondant à ce groupe de PCC est déterminée comme étant le polygone reliant des points de contour les plus proches du centroïde du groupe de PCC (voir la Figure 3.18). Plus précisément, une fois que les points dans S_2 sont regroupés, et pour chaque groupe, les auteurs extraient le centroïde P_{CI} des points dans ce groupe. Puis, les auteurs extraient des segments de points de contour P_c tel que pour chaque point $p \in P_c$: $distance(p, P_{CI}) \leq d_r$ (d_r est un seuil prédéfini, les auteurs fixent $d_r = 1{,}4w$). Ensuite, les auteurs extraient dans chaque segment P_c, le point P_z qui est plus proche du controïde P_{CI}. Le polygone dont les sommets correspondent aux points de P_z définit une zone ambiguë (voir la Figure 3.18).

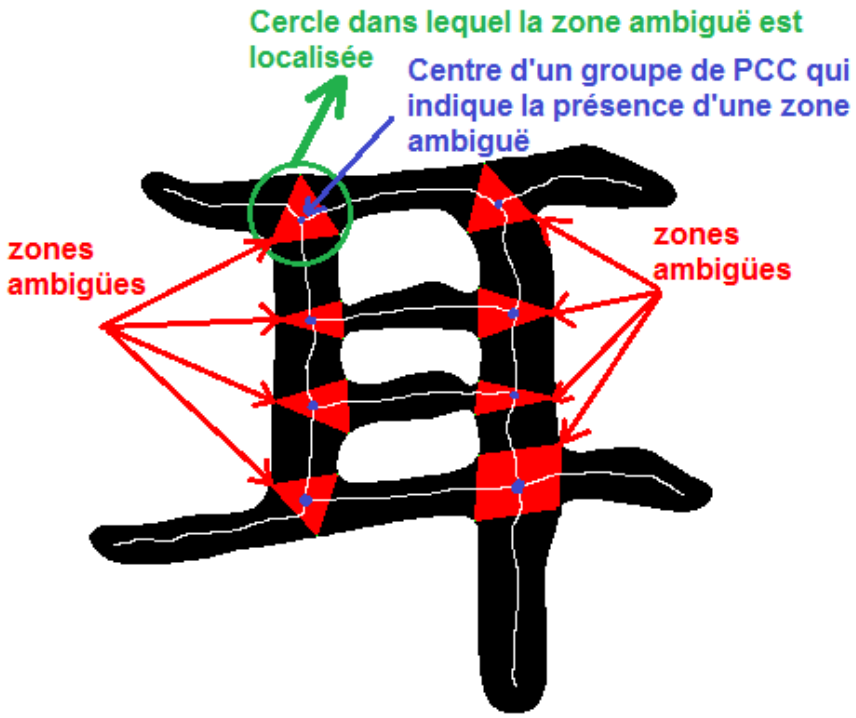


Figure 3.18 : Localisation des zones ambiguës selon [Su *et al.* 2009]

Conclusion :

Nous avons présenté dans cette section la méthode d'extraction de *strokes* primaires basée sur la détection de zones ambiguës [Su *et al.* 2009] que nous avons sélectionnée suite à une étude expérimentale menée sur 3 bases de caractéristiques très différentes. Cette méthode ne nécessite aucun apprentissage et peut donc s'adapter sur des documents multi-langages. Bien qu'elle repose sur de nombreux seuils fixés le plus souvent en fonction de la largeur moyenne des traits d'écriture, elle donne des résultats assez stables.

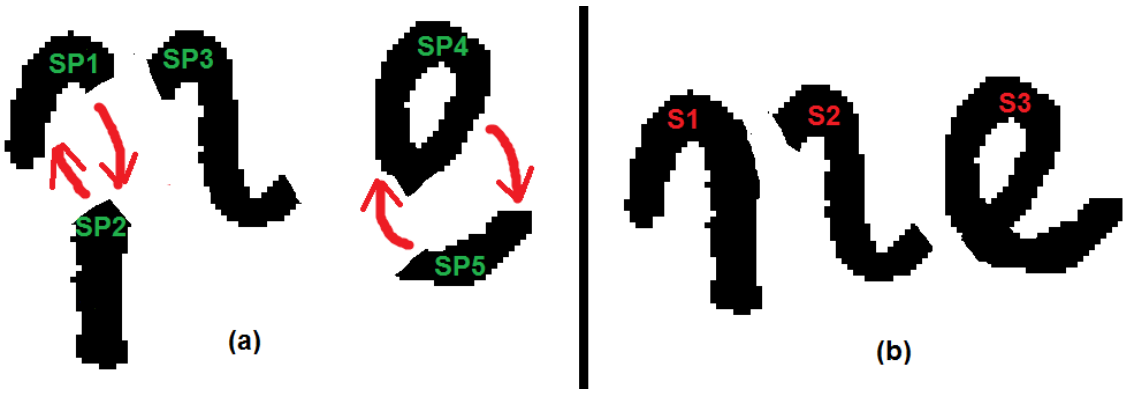


Figure 3.19 : Regroupement de strokes primaires. (a) : les strokes primaires avant le regroupement. (b) : les strokes résultant du regroupement

Une fois que nous avons appliqué la méthode de [Su *et al.* 2009] sur les composantes connexes extraits des images binarisées, nous définissons les « ***strokes* primaires** » comme l'ensemble des points connexes de l'écriture entre deux extrémités ou zones ambiguës. Après l'extraction de *strokes* primaires, les *strokes* primaires sont regroupés pour former les *strokes*. La Figure 3.19 montre un exemple : le *stroke* primaire SP1 est regroupé avec le *stroke* primaire SP2 pour former le *stroke* S1, le *stroke* primaire SP4 est regroupé avec le *stroke* primaire SP5 pour former le *stroke* S3 . Nous présentons dans la section ci-après, notre méthode de regroupement de *strokes* primaires.

3.3. Extraction de *strokes* par regroupement de *strokes* primaires

Lorsque les zones ambiguës ont été détectées et que les *strokes* primaires ont été extraits, il faut regrouper les *strokes* primaires en *strokes*. Les *strokes* sont donc des regroupements spatiaux de *strokes* primaires, donc de plus grande taille, se rapprochant des traits constituant l'écriture. Nous sommes ici sur les éléments de constitution de l'écriture, il est donc nécessaire que les *strokes* fassent sens par l'utilisateur. Pour regrouper les *strokes* primaires, essentiellement, les ambigüités dans les zones ambiguës doivent être résolues le plus possible. Dès lors, nous nous appuierons sur la « continuité » des *strokes*, cette continuité pouvait illustrer les traits composés par l'utilisateur, lors de sa requête. Dans la première partie de cette section, nous présentons quelques-unes des méthodes d'analyse de continuité de *strokes* primaires dans la littérature. Dans la deuxième partie, nous présentons notre méthode de regroupement de *strokes* primaires basée sur l'analyse de continuité que nous proposons.

3.3.1. Les méthodes d'analyse de continuité de pair de strokes primaires

Pour analyser la continuité d'une paire de *strokes* primaires joints à une même zone ambiguë, la plupart des approches dans la littérature sont basées sur la théorie de Gestalt, qui est une théorie psychologique inspirée du traitement visuel humain. Selon cette théorie, deux *strokes* primaires joints à une même zone ambiguë appartiennent à un même *stroke* (c'est-à-dire, ces deux *strokes* primaires sont continus) s'ils satisfont les 2 règles suivantes :

- ***La règle de largeur de section des strokes*** : les *strokes* sont composés de sections ayant approximativement la même largeur.

- ***La règle de bonne continuation*** : la courbure d'un même *stroke* ne peut pas brusquement changer.

Dans cette section, nous présentons quelques-unes des méthodes basées sur la théorie de Gestalt comme celles de [Liu *et al.* 1997], [Huang & Yasuhara 1995], [Jager 1996], [Qiao *et al.* 2006], [Plamondon & Privitera 1999] et [Su *et al.* 2009].

La méthode de [Liu *et al.* 1997] est basée sur l'image de squelette. Plus précisément, les auteurs extraient l'image du squelette pour détecter les zones ambiguës en utilisant le « critère de cercle maximal » (voir section 3.2.1.1). Puis, ils approximent l'image du squelette en polygones. Après l'approximation polygonale, les *strokes* primaires sont des segments linéaires et les zones ambiguës sont des nœuds qui connectent les *strokes* primaires. Pour analyser la continuité des paires des *strokes* primaires joints à une même zone ambiguë, les auteurs utilisent un ensemble de 4 règles qui dépendent du nombre de *strokes* primaires liées à la zone ambiguë, et éventuellement des angles entre ces *strokes* primaires réduits à des segments linéaires.

La méthode de [Liu *et al.* 1997] est simple et facile à implémenter. Mais l'inconvénient de cette méthode est qu'elle est conçue pour les documents chinois, dans lesquels la plupart des *strokes* sont des lignes droites et où l'écriture n'est pas très cursive. De plus, elle ne prend pas en compte la largeur de *stroke*, qui est un paramètre important dans l'analyse de continuité (règle de largeur de section des *strokes*).

Pour que la méthode de regroupement soit plus adaptée sur les écritures cursives, [Huang & Yasuhara 1995] propose la méthode d'approximation de courbe SLALOM pour calculer la fonction de continuité de 2 *strokes* primaires (approximés par des segments) joints à une même zone ambiguë. Plus précisément, étant donné $\{u_1, u_2, \dots, u_i, \dots, u_M\}$ une chaîne de points des deux segments s_1 et s_2 joints à une même zone ambiguë, les auteurs extraient une courbe approximative $\left(g_x(u), g_y(u)\right)$ qui minimise les valeurs $J_x(g_x)$ et $J_y(g_y)$ définies comme suit :

$$J_x(g_x) = \int \left(\frac{d^2}{du^2} g_x(u)\right)^2 du + \alpha \sum_{i=1}^{M} \left(g_x(u_i) - u_{i_x}\right)^2$$

$$J_y(g_y) = \int \left(\frac{d^2}{du^2} g_y(u)\right)^2 du + \alpha \sum_{i=1}^{M} \left(g_y(u_i) - u_{i_y}\right)^2$$

Où α est une constante prédéfinie, $\left(u_{i_x}, u_{i_y}\right)$ est la coordonnée d'un point u_i. En se basant sur les valeurs minimales trouvées (J_x^* et J_y^*), les auteurs définissent $C_s = -\left(J_x^* + J_y^*\right)$ comme valeur de continuité entre deux segments s_1 et s_2. Plus la valeur C_s est élevée, plus les deux segments s_1 et s_2 sont susceptibles d'être regroupés. Un inconvénient de la méthode de [Huang & Yasuhara 1995] est qu'elle ne prend pas en compte la largeur de *stroke* qui est un paramètre importante dans l'analyse de continuité (règle de largeur de section des *strokes*).

En fin, un autre inconvénient des méthodes de [Liu *et al.* 1997] et de [Huang & Yasuhara 1995] est que le calcul est basé sur l'image de squelette et donc qu'il est sensible au bruit ou aux distorsions. Pour contourner ce problème et au lieu de se baser sur l'image de squelette, [Qiao & Yasuhara 2004], [Plamondon & Privitera 1999], [Su *et al.* 2009] utilisent les pixels de l'image dans la zone de *stroke* primaire (voir la Figure 3.20).

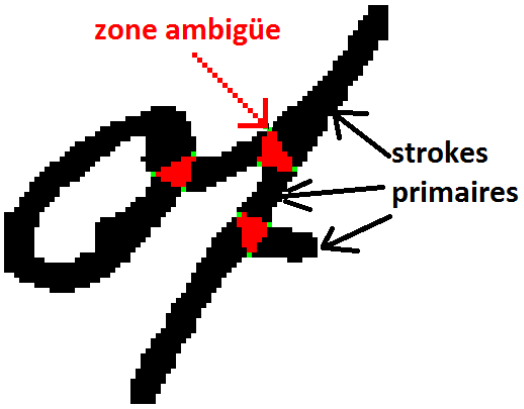


Figure 3.20 : Zone ambiguë et *strokes* primaires

[Plamondon & Privitera 1999] utilise une fenêtre glissante de gauche à droite pour détecter les zones ambiguë (voir section 3.2.1.2). Une fois qu'une zone ambiguë z_a est localisée, et pour chaque paire de *strokes* primaires S_i et S_j joints à z_a, les auteurs assignent un score de continuité Ω_{S_i,S_j} à cette paire de *strokes* primaires. Plus le score Ω_{S_i,S_j} est élevé, plus les deux *strokes* S_i et S_j sont susceptibles d'être regroupés. Le calcul du score Ω_{S_i,S_j} est basé sur la variation de l'épaisseur de *stroke* W_{S_i,S_j} et le dérivé de courbure maximal $\Psi^*_{S_i,S_j}$:

$$\Omega_{S_i,S_j} = \Psi^*_{S_i,S_j}\left(\alpha - \exp\left(\frac{-\left(d_i - d_j\right)^2}{\sigma^2}\right)\right)\eta$$

où α, σ et η $(0 < \eta < 1)$ sont des constantes prédéfinies. d_i est une estimation de l'épaisseur du *stroke* primaire S_i et d_j est une estimation de l'épaisseur du *stroke* primaire S_j. Pour plus de détail sur l'estimation de l'épaisseur d'un *stroke* primaire et de la dérivée de la courbure maximale, nous renvoyons le lecteur à [Plamondon & Privitera 1999].

La méthode de [Plamondon & Privitera 1999] utilise les pixels de l'image dans la zone de *stroke* primaire dans son calcul. Elle est donc plus stable en présence de bruits et de distorsions que les méthodes basées sur les squelettes. Elle prend en compte également la largeur de *stroke* comme paramètre de l'analyse. Néanmoins, un inconvénient de cette méthode est que l'on doit régler manuellement les paramètres σ, α et η. Il est donc difficile de fixer des valeurs de paramètres pour que le système puisse fonctionner avec des documents différents pour lesquels les critères de continuité varient de manière parfois importante (par exemple, les documents de différents langages, les documents de différentes époques, *etc.*). Afin de contourner ce problème [Qiao & Yasuhara 2004] et [Su *et al.* 2009] utilisent des techniques d'apprentissage pour que le système puisse apprendre les paramètres de continuité d'une paire de *strokes* primaires adapté pour une collection de documents donnée.

Plus précisément, [Qiao & Yasuhara 2004] calculent, pour chaque paire de *strokes* primaires connectés à une zone ambiguë, la courbure k. Les auteurs définissent ainsi la probabilité d'une paire de *strokes* primaire d'être continue comme $P_{ct} = e^{-mk}$ où m est une constante. Selon cette formule, plus la courbure k est petite, plus les deux *strokes* primaires sont susceptibles d'être regroupés (on parle de paire continue). Pour estimer la valeur m optimale, les auteurs utilisent l'algorithme d'estimation du maximum de vraisemblance avec une base d'apprentissage qui contient un grand nombre d'exemples incluant à la fois paires de *strokes* continues et paires de *strokes* discontinues. Pour estimer la courbure k, les auteurs se basent sur l'angle entre les directions principales des deux *strokes* primaires concernés. Plus précisément, supposons que S_i et S_j soient deux *strokes* primaires joints à une même zone ambiguë ; p_i et p_j sont respectivement les deux points de squelette dans S_i et S_j qui sont les plus proches de la zone ambiguë (voir Figure 3.21) ; d_{α_i} et d_{α_j} sont respectivement les deux directions principales de S_i et S_j ; α_i est l'angle formé par d_{α_i} et l'axe O_x ; α_j est l'angle formé par d_{α_j} et l'axe O_x ; β est l'angle formé par le vecteur $\overrightarrow{p_i p_j}$ et l'axe O_x. Les auteurs estiment la courbure k comme le changement de direction de S_i à S_j. La courbure k est donc calculée comme suit :

$$k = \left|\pi - |\alpha_i - \beta|\right| + \left|\alpha_j - \beta\right|$$

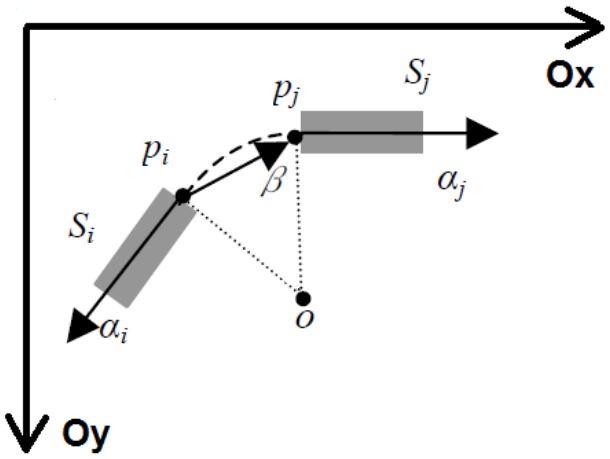


Figure 3.21 : Estimation de la courbure [Qiao & Yasuhara 2004]

Pour calculer la direction principale d_{α_i} d'un *stroke* primaire S_i, les auteurs sélectionnent une zone réduite de *stroke* qui est proche de la zone ambiguë (voir la Figure 3.22). La longueur de cette zone est estimée à $2w$, où w est l'épaisseur moyenne estimée des *strokes* de la collection. Puis, ils appliquent une ACP (Analyse en Composantes Principales) sur les coordonnées des pixels dans cette zone. La direction du premier axe principal de l'ACP correspond à la direction principale d_{α_i} du *stroke* primaire S_i.

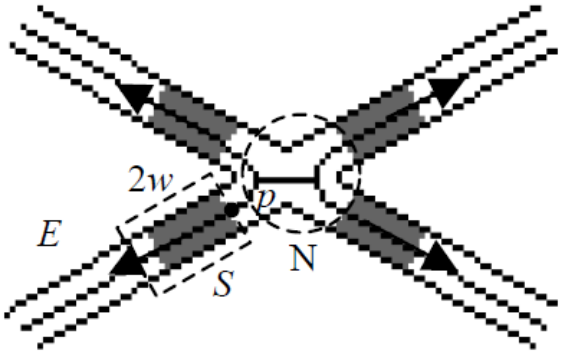


Figure 3.22 : Calcul de la direction principale d'un *stroke* primaire [Qiao & Yasuhara 2004]

L'inconvénient de la méthode de [Qiao & Yasuhara 2004] est qu'elle utilise seulement la courbure comme caractéristique pour analyser la continuité. Il ne prend pas en compte d'autres caractéristiques importantes comme par exemple la variation dans la largeur du *stroke*. Pour contourner ce problème, [Su *et al.* 2009] a introduit des autres caractéristiques dans son analyse de continuité comme :

- Le nombre de *strokes* primaires joints à la zone ambiguë
- La déviation de l'angle entre deux *strokes* primaires
- La différence de l'épaisseur entre deux *strokes* primaires
- La variation de courbure

La décision finale (continuité/discontinuité) est prise grâce à un classifieur Bayésien.

L'inconvénient des méthodes de [Qiao & Yasuhara 2004], [Su *et al.* 2009] est qu'elles reposent sur un apprentissage supervisé, et donc qu'elles nécessitent une annotation préalable d'une partie de la collection de documents à déduire, ce qui suppose que l'on ait utilisé dans la base de documents.

Quelques-unes des autres approches de la littérature sont basées sur une fonction de continuité comme la méthode de [Lallican 1997], [Nakajima *et al.* 1999]. [Lallican 1997] utilisent un filtre Kalman basé sur la courbure du trait et la position des points dans le trait comme paramètres afin de décider de la continuité de deux *strokes* primaires. [Nakajima *et al.* 1999] assimile le trait du *stroke* aux courbes paramétriques B-Spline. Les méthodes de ce type sont plus rapides que les méthodes basées sur la théorie de Gestalt, mais les *strokes* produits par ces méthodes sont moins significatifs de la composition de l'écriture, et généralement plus éloignés de la perception humaine.

Conclusion :

Nous avons présenté dans cette partie quelques-unes des méthodes d'analyse de continuité existant de la littérature. La plupart d'entre elles sont basées sur la théorie de Gestalt, qui est une théorie psychologique inspirée du traitement visuel de l'humain. Nous pouvons définir une taxonomie de ces méthodes qui distingue 2 familles : Les méthodes basées sur squelette et les méthodes basées sur tous les pixels des *strokes*. Généralement, les méthodes basées sur squelette sont faciles à implémenter et plus rapides que les méthodes basées sur tous pixels des *strokes*, mais elles ne sont pas stables en présence de bruits et de distorsions. Nous pouvons également catégoriser les méthodes d'analyse de continuité en 2 familles : les méthodes basées sur un apprentissage et les méthodes sans apprentissage. Les méthodes basées sur un apprentissage utilisent une base annotée pour que le système puisse apprendre les caractéristiques de la continuité d'une paire de *strokes* primaires pour une collection de documents donnée. Mais, de telles bases d'apprentissages sont couteuses à obtenir, et cela suppose une connaissance

préalable de chaque script / langage, dont nous ne disposons pas dans notre contexte applicatif.

En nous inspirant des méthodes de la littérature, nous proposons donc une méthode de regroupement de *strokes* primaires qui est adaptée à notre contexte. Nous présentons cette méthode dans la section ci-après.

3.3.2. Contribution 1 : méthode proposée de regroupement de strokes primaires

Après l'extraction de *strokes* primaires issus du traitement d'une même collection de documents, ces *strokes* primaires sont regroupés en « *strokes* » qui sont les entités de plus grande taille dont nous espérons qu'elles fassent sens pour un humain connaissant suffisamment le script et le langage utilisé pour juger de leur pertinence. Pour regrouper les *strokes* primaires, il faut dans un premier temps, et pour chaque paire de *strokes* primaires joints à une même zone ambiguë, appliquer une méthode d'analyse de continuité pour déterminer si ces deux *strokes* primaires sont continus (appartiennent à un même *stroke*), ou non. Dès lors, nous regroupons les *strokes* primaires qui ont été déterminés comme continus.

Nous présentons dans cette section la méthode proposée pour le regroupement de *strokes* primaires. Plus précisément, cette section consiste en 2 parties. Dans la première partie, nous présentons la méthode d'analyse de continuité de *strokes* primaires que nous proposons. Dans la deuxième partie, nous présentons la méthode de regroupement de *strokes* primaires proposée et nous l'illustrerons avec quelques exemples.

3.3.2.1. Analyse de continuité de strokes primaires

Notre objectif est de concevoir une approche capable de s'adapter avec des documents différents, en l'absence d'information à propos du script/langage utilisé (et donc d'annotation de la base). Nous ne pouvons donc pas utiliser une apprentissage supervisé pour analyser la continuité des *strokes* primaires. Pour cette raison, nous proposons une méthode d'analyse de continuité non-supervisé qui peut s'adapter à des documents de différents scripts/langages. Pour que les *strokes* regroupés soient le plus proches possible de la perception humaine, notre méthode d'analyse de continuité est basée sur la théorie de Gestalt qui est impliquée dans le traitement visuel humain de l'écriture. Comme évoqué plus haut, selon cette théorie, deux *strokes* primaires joints à une même zone ambiguë

appartiennent à un même *stroke* (c'est-à-dire, ces deux *strokes* primaires sont continus) s'ils satisfont les 2 règles suivantes :

- ***La règle de largeur de section des strokes*** : les *strokes* ont généralement des sections ayant approximativement la même largeur. (règle R1)
- ***La règle de bonne continuation*** : la courbure d'un même *stroke* ne peut pas brusquement changer. (règle R2)

Pour chaque paire de *strokes* primaires S_i et S_j joints à la zone ambiguë Z_a, nous générons un *stroke* hypothétique en regroupant S_i et S_j. Comme dans la méthode de [Su *et al.* 2009], nous analysons la variation de l'épaisseur de *stroke* et la variation de la courbure de *stroke*. Mais au lieu d'utiliser un apprentissage supervisé pour analyser la continuité des *strokes* comme [Su *et al.* 2009], nous utilisons un ensemble de seuils appliqués sur la variation de l'épaisseur de *stroke* et sur la variation de courbure de *stroke*. Si le *stroke* hypothétique généré satisfait aux règles R1 et R2 de la théorie de gestalt, S_i et S_j sont regroupés.

Comme pour [Su *et al.* 2009], l'analyse de la variation de l'épaisseur de *stroke* et l'analyse de la courbure de *stroke* ne sont pas basées sur l'image de squelette car elle n'est pas stable en présence de bruits et distorsions. Nous utilisons plutôt les pixels de l'image dans la zone de *stroke* primaire. Pour faire ces analyses, nous extrayons, pour chaque *stroke* primaire, la liste ordonnée des « Points d'échantillon de la Trajectoire de l'écriture » (PET) de ce *stroke* primaire. La liste des PET est au fait une forme réduite d'un *stroke* primaire, représentant la trajectoire de ce *stroke* primaire. Nous présentons l'algorithme d'extraction des PET ci-après :

Algorithme d'extraction des PET d'un *stroke* primaire

Supposons nous avons un *stroke* primaire S_i joint à la zone ambiguë Z_a. P_1 et P_2 sont 2 points opposés du contour de S_i. Le centre C de P_1 et P_2 est un point dans la liste de PET de ce *stroke* primaire. En traçant 2 points P_1 et P_2 sur le contour de S_i, nous pouvons obtenir la liste de PET de S_i. Plus précisément, nous appliquons l'algorithme suivant :

- *Étape 1* : PET = {∅}
- *Étape 2* : Prendre le point P_1 dans le contour de S_i proche de la zone ambiguë Z_a.
- *Étape 3* : Trouver le point P_2 dans le contour de S_i, opposé de P_1.
- *Étape 4* : Répéter jusqu'à ce que tous les points du contour soient tracés :
    - *Étape 4.1* : Trouver le centre C de P_1 et P_2. Ajouter C à PET
    - *Étape 4.2* : Supposons que le contour d'un *stroke* primaire est une liste ordonnée de points. Nous utilisons les définitions suivantes :

- $B(p,n)$ est le $n^{ième}$ point prédécesseur du point p dans le contour.
- $F(p,n)$ est le $n^{ième}$ point successeur du point p dans le contour.

- *Étape 4.3* : Localiser les points : $P'_1 = B(P_1, d_s)$ et $P'_2 = F(P_2, d_s)$ où d_s est la valeur entier de la distance euclidienne entre deux points P_1 et P_2.
- *Étape 4.4* : Calculer les valeurs suivantes :
  - $d(P'_1, P'_2)$: la distance euclidienne entre P'_1 et P'_2
  - $d(P'_1, P_2)$: la distance euclidienne entre P'_1 et P_2
  - $d(P_1, P'_2)$: la distance euclidienne entre P_1 et P'_2
- *Étape 4.5* : Si $d(P'_1, P'_2) \leq d(P'_1, P_2)$ et $d(P'_1, P'_2) \leq d(P_1, P'_2)$ alors nous déplaçons P_1 et P_2 vers P'_1 et P'_2 : $P_1 = P'_1$, $P_2 = P'_2$. Retourner à l'étape 4.1. Sinon, aller à l'étape 4.6.
- *Étape 4.6* : Si $d(P'_1, P_2) \leq d(P'_1, P'_2)$ et $d(P'_1, P_2) \leq d(P_1, P'_2)$ alors nous déplaçons P_1 vers P'_1 : $P_1 = P'_1$. Retourner à l'étape 4.1. Sinon : aller à l'étape 4.7.
- *Étape 4.7* : Si $d(P_1, P'_2) \leq d(P'_1, P'_2)$ et $d(P_1, P'_2) \leq d(P'_1, P_2)$ alors nous déplaçons P_2 vers P'_2 : $P_2 = P'_2$. Retourner à l'étape 4.1.

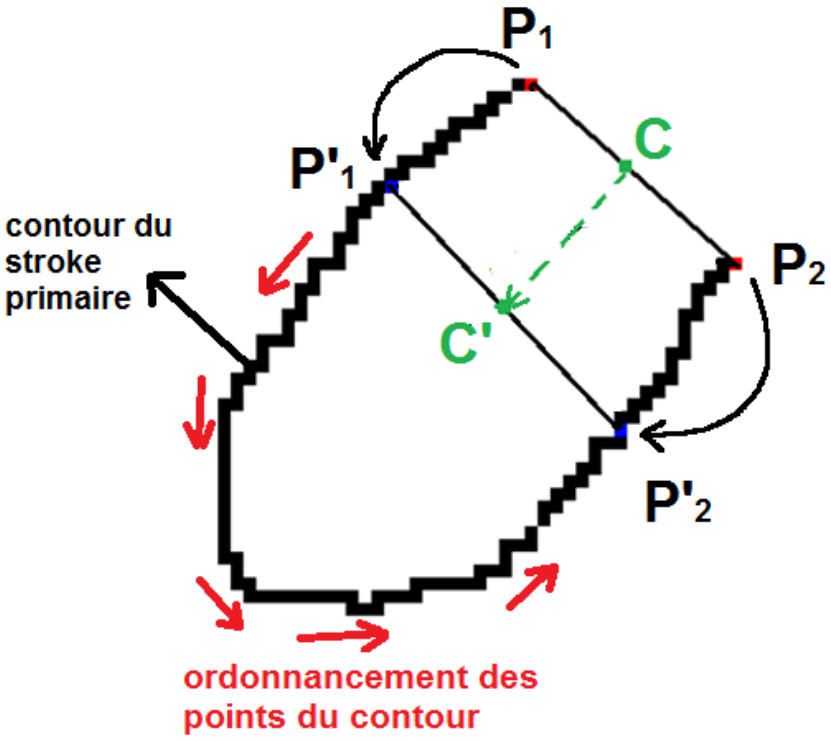


Figure 3.23 : Extraction des Points de Trajectoire

Les PET d'un S_i sont les chaines de points : $\{p_0, p_1, \ldots, p_n\}$. Le point p_0 d'une chaine est le point le plus proche d'une zone ambiguë Z_a. En réalité, nous n'utilisons pas

tous les points de la chaine, mais une partie de la chaine qui s'appelle *chaine de support* (voir Figure 3.24). La raison pour laquelle nous n'utilisons pas la chaine complète, c'est que la trajectoire peut changer sa direction après un croisement. Nous voulons calculer la continuité de *stroke*. Ainsi, nous nous intéressons à la chaîne de points depuis le croisement jusqu'à l'endroit où la direction change de façon importante (voir Figure 3.24). L'algorithme pour déterminer la chaine de support est présenté ci-après :

Algorithme pour déterminer la chaine de support :

Pour déterminer la chaine de support d'une chaine $C_n = \{p_0, p_1, \dots, p_n\}$, nous utilisons l'algorithme suivant :

- *Étape 1* : $g = 2$
- *Étape 2* : Trouver la direction principale de la chaine $C_g = \{p_0, p_1, \dots, p_g\} : D_g$. D_g est définie comme le premier composante principal obtenu en appliquant l'Analyse en Composantes Principales (ACP) sur les coordonnées des points de chaine C_g
- *Étape 3* : Trouver la direction principale de la chaine $C_{g+1} = \{p_0, p_1, \dots, p_{g+1}\} : D_{g+1}$. Supposons que $E(D_g, D_{g+1})$ est l'angle minimale formée par D_g et D_{g+1}. Si $E(D_g, D_{g+1}) > \phi$ alors $g^* = g$; Retourner la chaine de support : $C_{g^*} = \{p_0, p_1, \dots, p_{g^*}\}$. Sinon, aller à l'étape 4.
- *Étape 4* : $g = g + 1$. Retourner à l'étape 2.

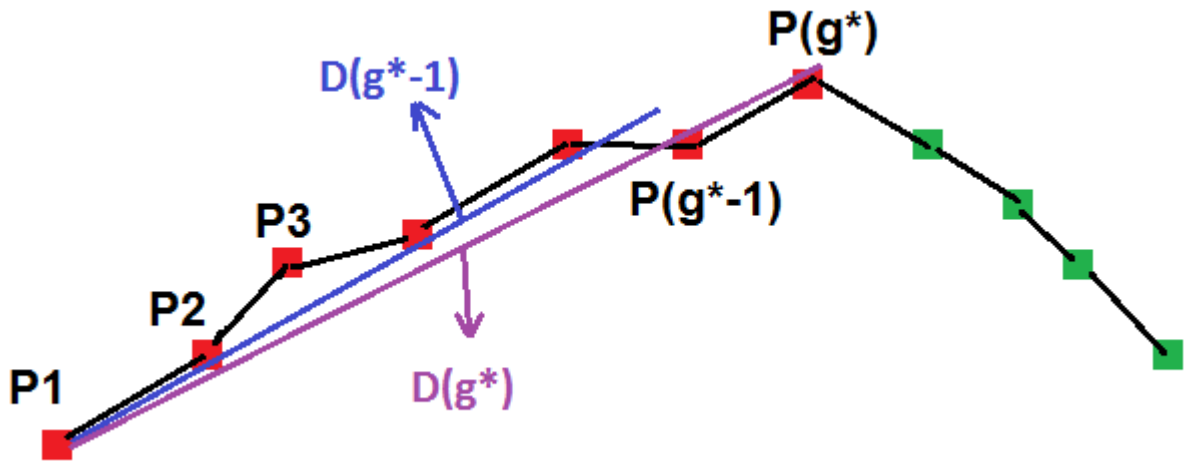


Figure 3.24 : Détermination d'une chaîne de support

Ici, ϕ est un seuil prédéfini qui indique la direction de la chaîne de support. Dans notre système, nous fixons $\phi = \pi/10$ en nous basant sur un ajustement réalisé sur une base de validation composée de documents écrits en plusieurs langages.

Après l'extraction des PET de 2 *strokes* primaires S_i et S_j joints à une zone ambiguë Z_a, nous appliquons une procédure en deux étapes pour analyser la continuité de S_i et S_j :

- ***Étape 1*** : ***Analyse de la variation de l'épaisseur de stroke***

Nous estimons les épaisseurs : w_i et w_j des *strokes* primaires S_i et S_j. Pour estimer l'épaisseur w_i du *stroke* primaire S_i, nous appliquons la méthode suivante :

- Nous calculons la direction principale $D(g_i^*)$ de la chaîne de PET de S_i. La direction principale $D(g_i^*)$ est calculée en appliquant une ACP sur les coordonnées des points de la chaîne. $D(g_i^*)$ est la direction du premier axe principal.
- Supposons que P_1 et P_2 soient les deux points opposés dans le contour de S_i qui sont les plus proches de la zone ambiguë (voir Figure 3.25). L'épaisseur w_i du *stroke* primaire S_i est estimée comme suite :

$$w_i = d(P_1, P_2) * \left|\sin\left(E\left(\mathrm{V}(P_1, P_2), D(g_i^*)\right)\right)\right|$$

Où $d(P_1, P_2)$ est la distance entre deux points P_1 et P_2, $\mathrm{V}(P_1, P_2)$ est la direction du vecteur $\overrightarrow{P_1, P_2}$, $E\left(\mathrm{V}(P_1, P_2), D(g_i^*)\right)$ est l'angle minimale formée par $\mathrm{V}(P_1, P_2)$ et $D(g_i^*)$.

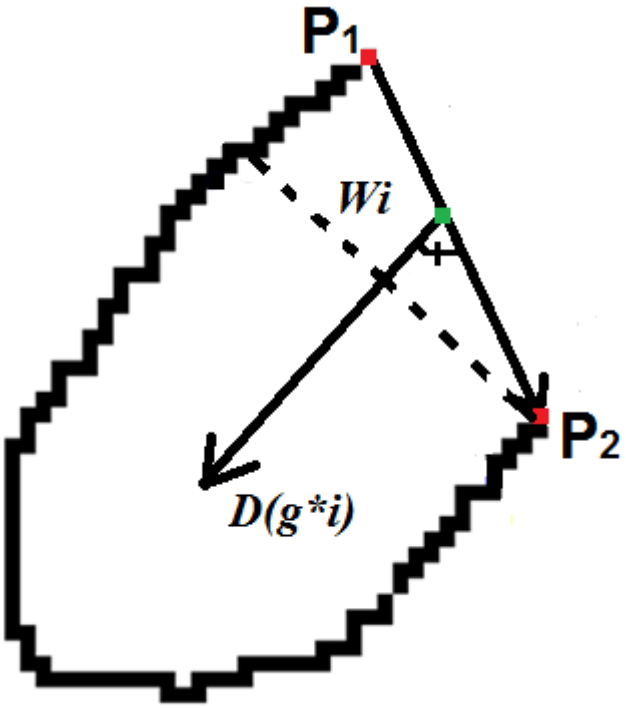


Figure 3.25 : Estimation de l'épaisseur d'un *stroke* primaire

La mesure $w_{ij} = \frac{|w_i - w_j|}{\max(w_i, w_j)}$ décrit la variation de l'épaisseur entre les *strokes* primaires S_i et S_j. Selon la règle R1 (**règle de largeur de section des *strokes***), si w_{ij} dépasse un seuil w_θ, S_i et S_j ne peuvent pas être regroupés. Ces deux *strokes* primaires appartiennent donc à deux *strokes* différents. Dans notre application, nous fixons le seuil $w_\theta = 0.3$ suite à un ajustement sur une base de validation composée de documents de différents langages.

- ***Étape 2*** : ***Analyse de la variation de courbure de stroke***

Après l'analyse de la variation de l'épaisseur de *stroke* et l'élimination des paires de *strokes* non regroupés, pour chaque paire de *strokes* primaires restantes, nous faisons l'analyse de la variation de la courbure de *stroke*. Supposons que $(\{p_{i,0}, p_{i,1}, \dots, p_{i,g_i^*}\}, g_i^* \geq 2)$ et $(\{p_{j,0}, p_{j,1}, \dots, p_{j,g_j^*}\}, g_j^* \geq 2)$ sont respectivement les PET de deux *strokes* primaires S_i et S_j joints à une même zone ambiguë Z_a, nous estimons la variation de courbure de *stroke* γ_{ij} de la manière suivante :

$$\gamma_{ij} = \frac{1}{g_i^* + g_j^* - 1} \sum_{k=-g_j^*}^{g_i^*-2} \frac{|\zeta_{i,k} - \zeta_{i,k+1}|}{d(p_{i,k}, p_{i,k+1})}$$

Où $d(p_{i,k}, p_{i,k+1})$ est la distance entre le point $p_{i,k}$ et le point $p_{i,k+1}$; $p_{i,-1} = p_{j,0}$; $p_{i,-2} = p_{j,1}$; … ; $p_{i,-g_j^*-1} = p_{j,g_j^*}$; $\zeta_{i,k}$ est la courbure estimée au point $p_{i,k}$:

$$\zeta_{i,k} = \frac{E\left(V(p_{i,k-1}, p_{i,k}), V(p_{i,k}, p_{i,k+1})\right)}{d(c_{i,k-1}, c_{i,k})}$$

Où $c_{i,k}$ est le point au centre de la ligne entre $p_{i,k}$ et $p_{(i,k+1)}$; $c_{i,k-1}$ est le point au centre de la ligne entre $p_{i,k-1}$ et $p_{(i,k)}$; $d(c_{i,k-1}, c_{i,k})$ est la distance entre le point $c_{i,k-1}$ et le point $c_{i,k}$; $V(p_{i,k}, p_{i,k+1})$ est la direction du vecteur $\overrightarrow{p_{i,k}p_{i,k+1}}$; $V(p_{i,k-1}, p_{i,k})$ est la direction du vecteur $\overrightarrow{p_{i,k-1}, p_{i,k}}$. $E\left(V(p_{i,k-1}, p_{i,k}), V(p_{i,k}, p_{i,k+1})\right)$ est l'angle minimal formé par $V(p_{i,k}, p_{i,k+1})$ et $V(p_{i,k-1}, p_{i,k})$.

Selon la règle R2 (**règle de bonne continuation**), les *strokes* primaires S_i et S_j sont regroupés si et seulement si γ_{ij} ne dépasse pas un seuil prédéfini γ_θ. Dans notre application, nous fixons $\gamma_\theta = 0{,}15$ suite à un ajustement sur une base de validation composée de documents de différents langages.

3.3.2.2. Regroupement de strokes primaires

Après l'analyse de la continuité de chaque paire de *strokes* primaires joints à une même zone ambiguë, nous regroupons successivement les couples de *strokes* primaires qui ont été déterminés comme continus. Un *stroke* est un groupe de *strokes* primaires continus.

Nous présentons dans cette section quelques exemples du regroupement de *strokes* primaires que nous réalisons :

Exemple 1 :

La Figure 3.26 montre une image du mot « cum » (extraite de la base « Saint Gall »)[1] avec ses zones ambiguës détectées (en rouge) et ses *strokes* primaires (en noir). Dans cet exemple, toutes les zones ambiguës sont détectées.

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/saint-gall-database

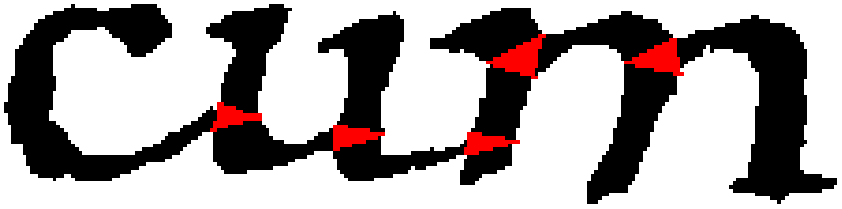


Figure 3.26 : Une image de mot avec ses zones ambiguës détectées

La Figure 3.27 et la Figure 3.28 montrent le résultat du regroupement des *strokes* primaires. Dans la Figure 3.27, si deux *strokes* primaires ont été déterminés comme continus, ils sont connectés par une ligne bleue. Les chaines grises représentent les chaînes de PET des *strokes* primaires.

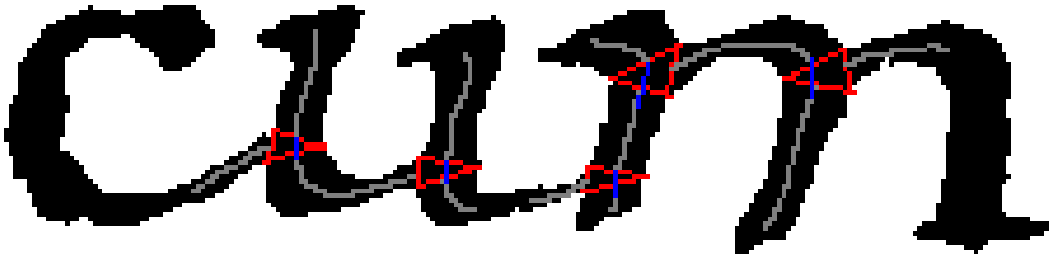


Figure 3.27 : Résultat du regroupement des *strokes* primaires

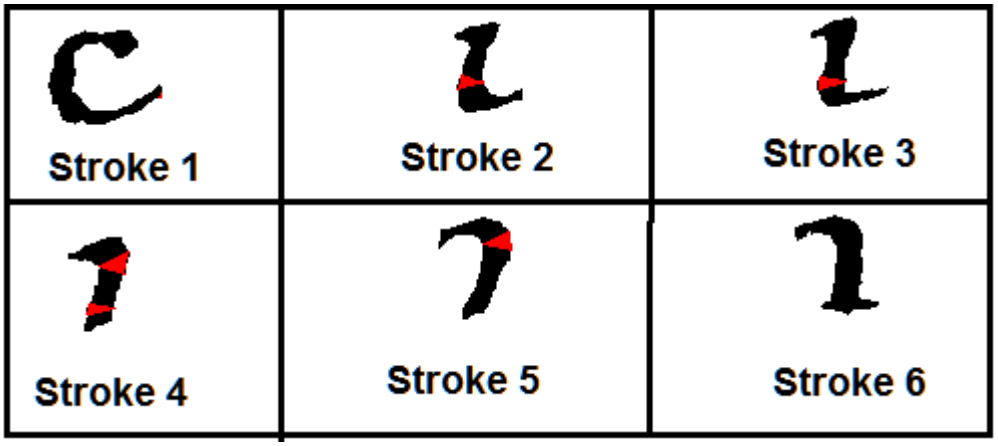


Figure 3.28 : *Strokes* extraits

La Figure 3.28 montre des *strokes* extraits. Tous les *strokes* nous semblent avoir un intérêt pour la composition de requêtes par un humain.

Exemple 2 :

La Figure 3.29 montre une image du mot « accepta » (extraite de la base « Saint Gall »[1]) avec ses zones ambiguës détectées (en rouge) et ses *strokes* primaires (en noir). Dans cet exemple, il y a une zone ambiguë qui n'est pas détectée.

Figure 3.29 : Une image de mot avec ses zones ambiguës détectées

La Figure 3.30 et la Figure 3.31 montrent le résultat du regroupement des *strokes* primaires. Dans la Figure 3.30, si deux *strokes* primaires ont été déterminés comme continus, ils sont connectés par une ligne bleue. Les chaînes grises représentent les chaînes de PET des *strokes* primaires.

Figure 3.30 : Résultat du regroupement des *strokes* primaires

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/saint-gall-database

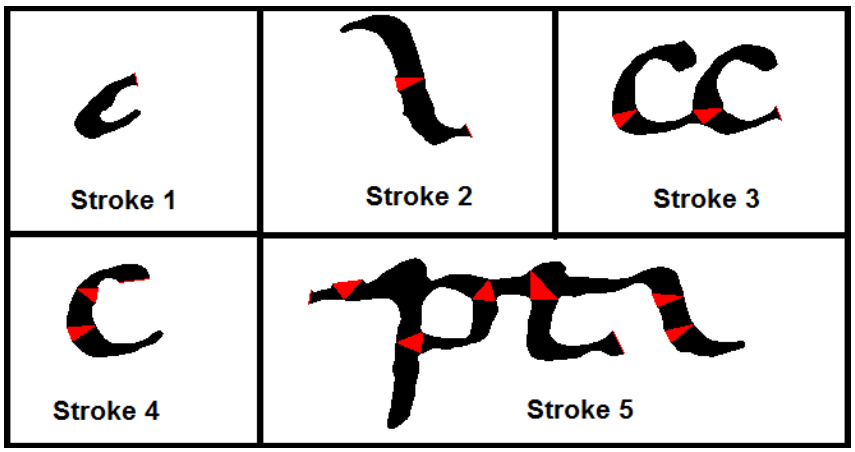


Figure 3.31 : *Strokes* extraits

La Figure 3.31 montre les *strokes* extraits. Les *strokes* 1, 2, 4 nous semblent avoir un intérêt pour la composition de requêtes par un humain. À cause d'une zone ambiguë non détectée et des mauvais résultats dans l'analyse de continuité, le *stroke* 5 est trop grand pour avoir de l'intérêt.

Exemple 3 :

La Figure 3.32 montre une image d'un mot chinois (extrait de la base « Chinois » - voir section 3.2.2.1) avec ses zones ambiguës détectées (en rouge) et ses *strokes* primaires (en noir). Dans cet exemple, toutes les zones ambiguës sont bien détectées.

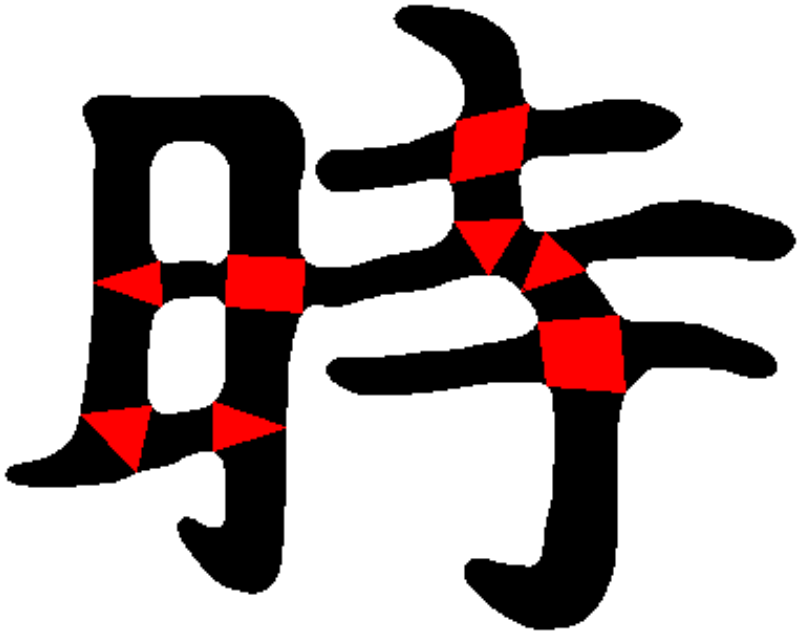


Figure 3.32 : Une image d'un mot chinois

La Figure 3.33 et la Figure 3.34 montrent le résultat du regroupement des *strokes* primaires. Dans la Figure 3.33, si deux *strokes* primaires ont été déterminés comme continus, ils sont connectés par une ligne bleue. Les chaînes grises représentent les chaînes de PET des *strokes* primaires.

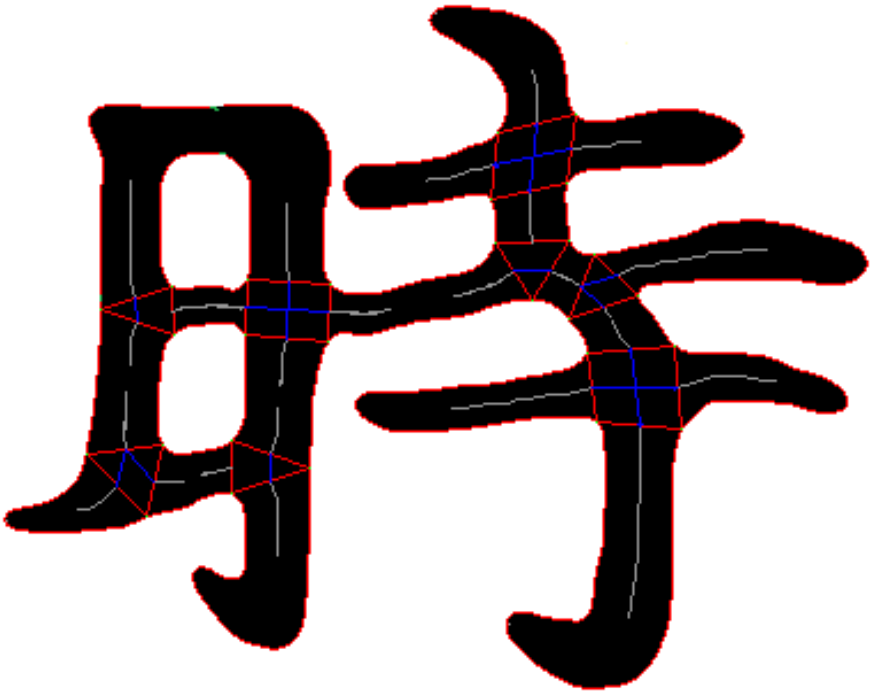


Figure 3.33 : Résultat du regroupement des *strokes* primaires

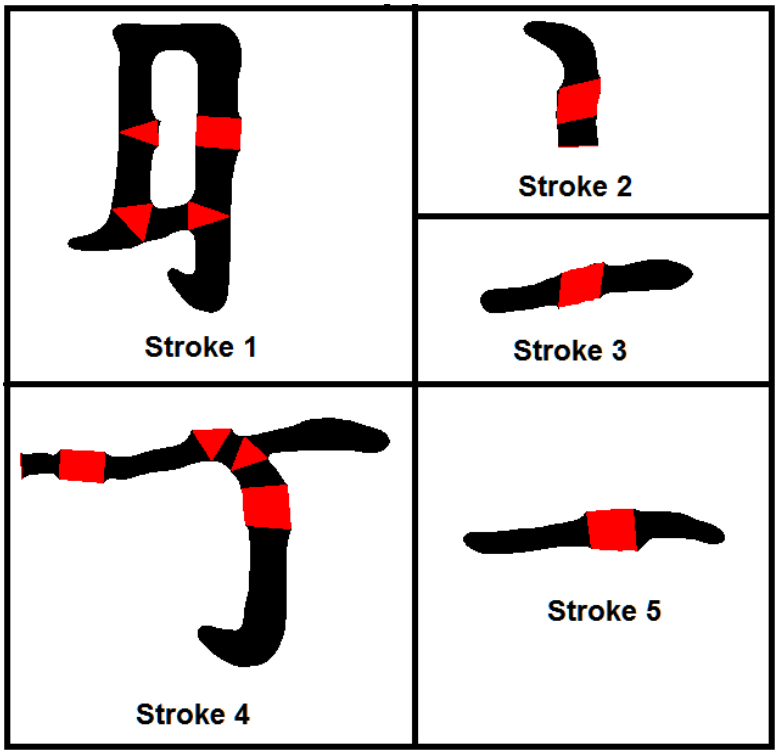


Figure 3.34 : *Strokes* extraits

La Figure 3.34 montre les *strokes* extraits. Les *strokes* 1, 2, 3, 5 nous semble avoir un intérêt pour la composition de requêtes par un humain. Le *stroke* 1 n'a pas d'intérêt pour la composition de requêtes.

Exemple 4 :

La Figure 3.35 montre une image d'un mot extrait de la base « Parzival » [1] avec ses zones ambiguës détectées (en rouge) et ses *strokes* primaires (en noir). Dans cet exemple, il y a une zone ambiguë qui est mal détectée.

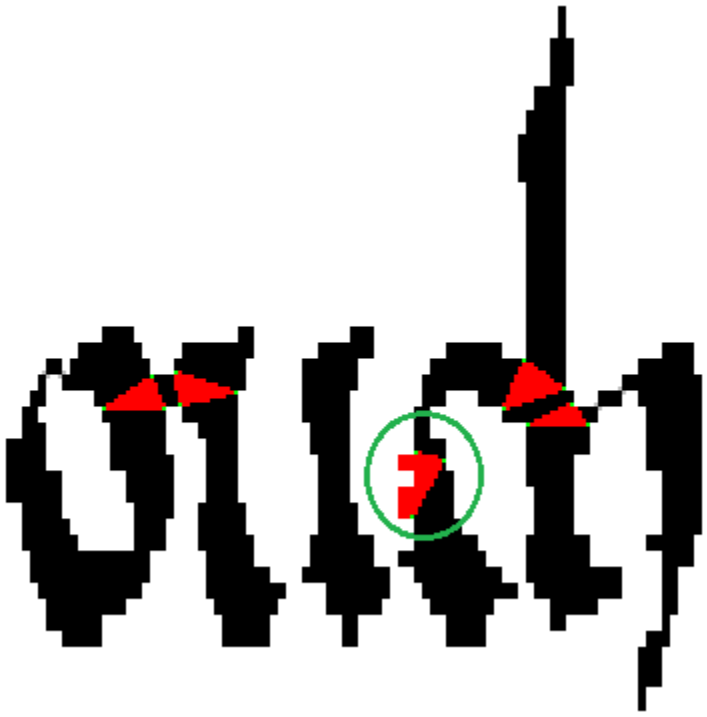


Figure 3.35 : Une image de mot extraite de la base « Parzival » avec ses zones ambiguës détectées

La Figure 3.36 et la Figure 3.37 montrent le résultat du regroupement des *strokes* primaires. Dans la Figure 3.36, si deux *strokes* primaires ont été déterminés comme continus, ils sont connectés par une ligne bleue. Les chaînes grises représentent les chaînes de PET des *strokes* primaires.

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/parzival-database

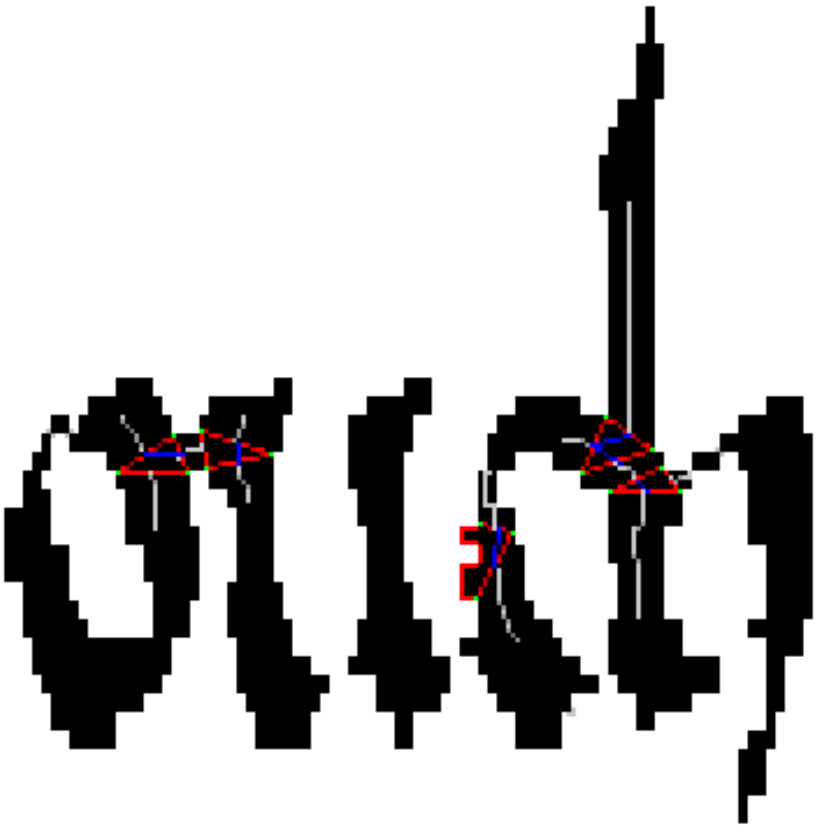


Figure 3.36 : Résultat du regroupement des *strokes* primaires

La Figure 3.37 montre les *strokes* extraits. Le *stroke* 1 et *stroke* 4 nous semble avoir un intérêt pour la composition de requêtes par un humain tandis que les *strokes* 2, 3 et 5 n'ont pas d'intérêt.

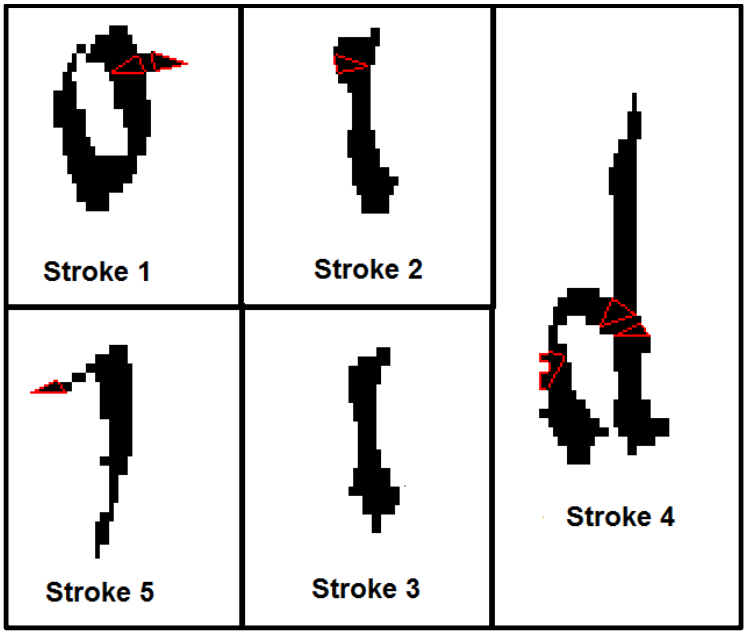


Figure 3.37 : *Strokes* extraits

Exemple 5 :

La Figure 3.38 montre une image extraite de la base « Washington » [1] avec ses zones ambiguës (en rouge) et ses *strokes* primaires (en noir). Dans cet exemple, il y a 3 zones ambiguës qui sont mal détectés. (À cause de la qualité médiocre de l'image binarisée).

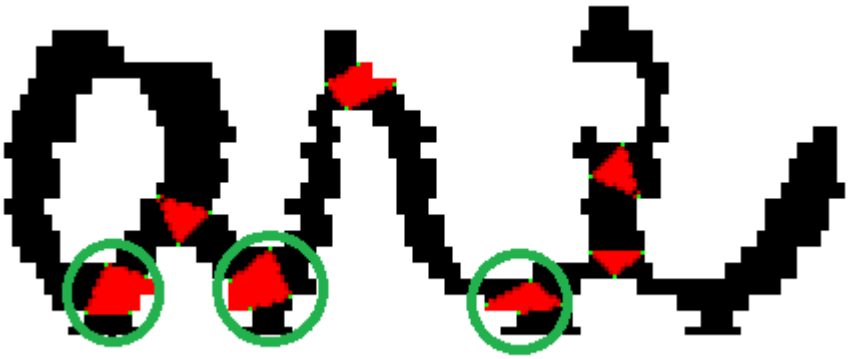


Figure 3.38 : Une image extraite de la base « Washington » avec ses zones ambiguës détectées

La Figure 3.39 et la Figure 3.40 montre le résultat de regroupement des *strokes* primaires. Dans la Figure 3.39 , si deux *strokes* primaires ont été déterminés comme continus, ils sont connectés par une ligne bleue. Les chaînes grises représentent les chaînes de PET des *strokes* primaires.

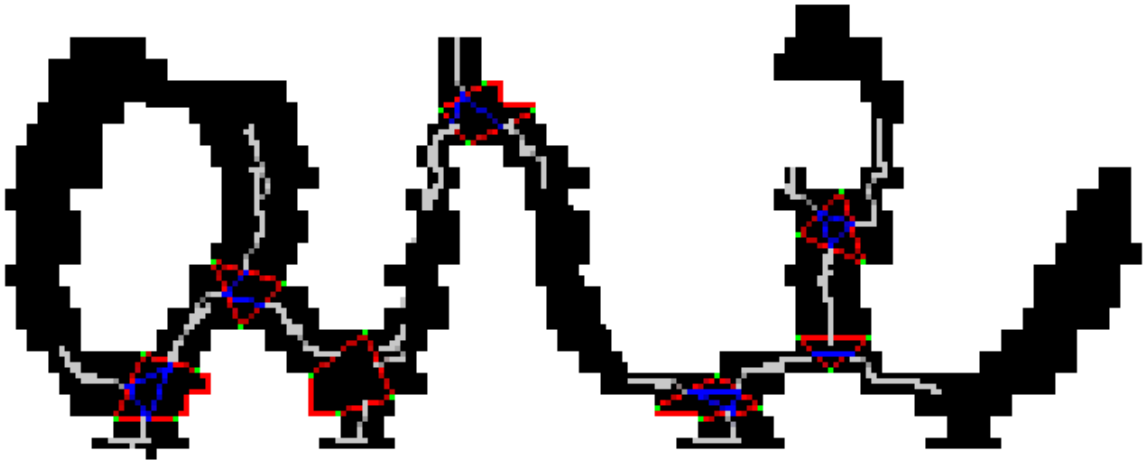


Figure 3.39 : Résultat du regroupement de *strokes* primaires

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/washington-database

La Figure 3.40 montre les *strokes* extraits. Le *stroke* 1 nous semble avoir un intérêt pour la composition de requête par un humain tandis que le *stroke* 3 et le *stroke* 4 n'ont pas d'intérêt.

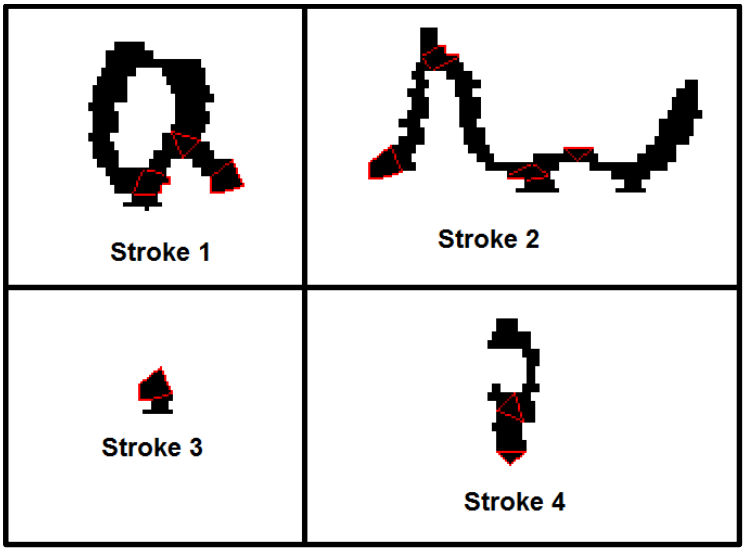


Figure 3.40 : *Strokes* extraits

Discussion :

Nous avons présenté dans cette section notre méthode de regroupement de *strokes* primaires avec quelques exemples de résultats obtenus. L'une de nos contributions principales est la conception d'une méthode d'analyse de continuité, basée sur la théorie de Gestalt, en utilisant une fonction de décision pour décider de si deux *strokes* primaires reliés par une zone ambiguë sont continus ou non.

À ce stade de la présentation de nos travaux, en l'absence de vérité-terrain sur ce que doivent être les invariants, nous ne pouvons pas fournir d'évaluation quantitative des résultats. Cependant, et en particulier en présence de documents particulièrement dégradés, les résultats produits par notre méthode sont difficilement exploitable pour un humain souhaitant composer des requêtes. Il en est de même pour la plupart des méthodes de la littérature. La cause majeure des mauvais résultats vient d'un problème intrinsèque de notre méthode qui applique une fonction de décision non supervisée pour décider de la continuité (ou non) de 2 *strokes* primaires joints à une même zone ambiguë. Notre méthode peut s'adapter à des documents de différents types sans connaissance préalable sur ces documents, mais en contrepartie, il est difficile d'en ajuster les seuils de manière omni-langage. Pour résoudre ce problème, nous proposons une méthode de raffinement interactif des *strokes* produits, qui sera présentée dans le chapitre 5.

Nous pourrions également envisager une initialisation des seuils qui se fasse sans information spécifique au langage utilisé, mais par ajustement sur une base de documents écrits avec le même type de langage (alphabétique, basé sur des logogrammes).

3.4. Contribution 2 : Méthode proposée pour l'extraction d'invariants par *clustering* de *strokes*

Dans les sections 3.2 et 3.3, nous avons présenté notre méthode pour extraire les *strokes* à partir du document. Dans cette section, nous présentons notre méthode d'extraction d'invariants par *clustering* de *strokes*.

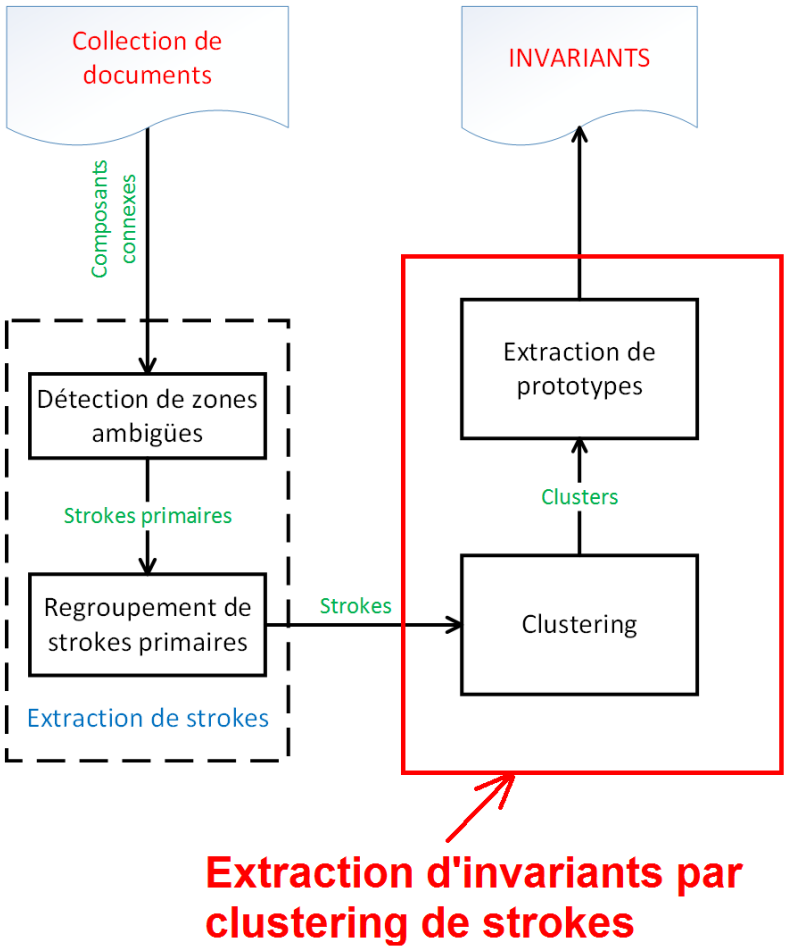


Figure 3.41 : Extraction d'invariants par *clustering* de *strokes*

Pour obtenir les invariants à partir du document, on applique un processus en 2 étapes. Premièrement, les *strokes* sont extraits en utilisant les méthodes de

détection de zones ambiguës et de regroupement de *strokes* primaires décrites ci-avant. Puis, les *strokes* sont regroupés en *clusters* en utilisant la méthode de *clustering* décrite ci-après. L'objectif est ici de regrouper les formes similaires en supposant qu'elles ont un sens pour l'utilisateur. Après le *clustering*, pour chaque *cluster*, nous trouvons le prototype de ce *cluster*. Les invariants sont définis comme étant ces prototypes.

Plus précisément, la méthode de *clustering* de *strokes* que nous proposons consiste en 4 étapes. L'étape 1 (l'étape de pré-*clustering*) est un filtre dans lequel nous appliquons une méthode *clustering* en utilisant des caractéristiques simples pour regrouper les *strokes* dans quelques méta-*clusters*. Dans l'étape 2 (l'étape de ré-*clustering*), pour chaque méta-*cluster* obtenu, nous appliquons plusieurs méthodes de *clustering* en utilisant des caractéristiques plus complexes pour obtenir des *clusters* de *strokes* plus représentatifs. Dans l'étape 3, à partir du résultat de l'étape 2, nous appliquons une méthode de consensus *clustering* en utilisant la mesure NMI_{max} [Vinh 2010a]. Après le consensus *clustering*, dans l'étape 4, nous choisissons pour chaque *cluster* de *strokes*, le prototype de ce *cluster*. Les invariants extraits sont définis comme étant ces prototypes.

Nous ne souhaitons pas d'invariance à l'échelle ni à la rotation dans le résultat de *clustering*. En effet et par exemple, pour le script en Latin, des caractères « n » peuvent être regroupés avec des caractères « u », des caractères en majuscules peuvent être regroupés avec des caractères en minuscules, *etc.* Pour atteindre cet objectif, dans la première étape, nous appliquons un algorithme de *clustering* sur tous les *strokes* dans la base de données en utilisant des descripteurs non-invariants à l'échelle pour regrouper les *strokes* de même taille dans des méta-clusters. Ensuite, dans l'étape 2, pour chaque méta-*cluster* obtenu, nous appliquons des algorithmes de *clustering* sur les *strokes* de ce *cluster* en utilisant des descripteurs non-invariants à la rotation.

Dans les sections suivantes, nous présentons les étapes dans notre méthode d'extraction d'invariants.

3.4.1. Première étape : pré-clustering

Cette étape consiste en un pré-*clustering* qui a pour objectif de regrouper les *strokes* de même taille. Nous utilisons comme descripteurs la largeur et la hauteur de la boîte englobant des *strokes* et nous appliquons un algorithme de *clustering* sur tous les *strokes* de la base pour regrouper les *strokes*. Après le *clustering*, les *strokes* de même taille sont regroupés dans un même méta-*cluster*.

Pour choisir la méthode de *clustering* la plus adaptée dans cette étape, nous réalisons une expérimentation avec 4 méthodes de *clusterings* : k-moyenne, DBScan, CAH et SOM (voir l'annexe B) et avec 3 bases de test différentes :

- ***Base Saint Gall*** : La base « Saint Gall » [1] contient des documents historiques manuscrits du langage Latin, écrits par un seul scripteur au 9ième siècle. Cette base contient 60 pages avec 11.597 images de mots. Dans cette expérimentation, nous prenons aléatoirement 20 pages de la base Saint Gall.
- ***Base Parzival*** : La base « Parzival » [2] contient les documents historiques manuscrits en langage allemand médiéval, écrits par 3 scripteurs au 13ième siècle. Cette base contient 47 pages avec 23.478 images de mots. Dans cette expérimentation, nous prenons aléatoirement 16 pages de la base Parzival.
- ***Base Washington*** : La base « Washington » [3] contient des documents historiques manuscrits en langage anglais, écrits par 2 scripteurs au 18ième siècle. Cette base contient 20 pages avec 4.894 images de mots. Dans cette expérimentation, nous prenons aléatoirement 7 pages de la base Washington complète.

Pour chaque base, nous appliquons notre méthode d'extraction de *strokes* sur les images de mots préalablement binarisées dans la base. Lorsque les *strokes* sont extraits, nous calculons, pour chaque *stroke*, la hauteur et la largeur du *stroke*. Puis, nous appliquons une normalisation sur les caractéristiques pour que les valeurs soient comprises entre 0 et 1. Ensuite, nous appliquons une méthode de *clustering* pour regrouper les *strokes* en méta-*clusters*. Nous souhaitons obtenir 4 méta-clusters : *strokes* peu longs et étroits, *strokes* longs et étroits, *strokes* peu longs et larges, *strokes* longs et larges. Nous réalisons 4 méthodes de *clustering* dans cette expérimentation :

- Méthode K-moyennes (voir annexe B.1) : La méthode des K-moyennes est représentatives des méthodes de *clustering* de partitionnement. Dans les méthodes de *clustering* de partitionnement, le nombre de clusters est habituellement prédéfini. Les clusters sont linéairement séparables. Parmi ces méthodes, la méthode K-moyennes est certainement la plus connue et la plus utilisée, notamment sur sa simplicité et son efficacité. En revanche, la méthode des K-moyennes est sensible à l'initialisation des k premières moyennes et le résultat dépend de la valeur de k. Dans cette expérimentation, nous choisissons k = 4.

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/saint-gall-database
[2] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/parzival-database
[3] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database/washington-database

- Méthode DBScan (voir annexe B.3) : Cette méthode est représentatives des méthodes basées sur la densité des données. Les méthodes basées sur la densité visent à partitionner les données en basant sur la distribution locale de ces données, et un groupe des données qui sont localement denses est considéré comme un *cluster*. La méthode de *clustering* DBScan est robuste aux données aberrantes. En revanche, ses paramètres sont difficiles à ajuster. De plus. Dans cette expérimentation, nous fixons $\epsilon = 0.0087$ et $n = 80$ pour tous les bases de données. (valeur ajusté heuristiquement sur l'ensemble des bases).
- Méthode CAH (voir annexe B.2) : Cette méthode est représentative des méthodes de *clustering* hiérarchiques. Les méthodes hiérarchiques fournissent une décomposition hiérarchique des clusters en sous-clusters. Les clusters ne sont pas nécessairement linéairement séparables. La méthode de *clustering* CAH construit les clusters par agrégations successives d'objets. L'arbre construit par l'algorithme CAH est déterministe car cet algorithme n'a pas besoin d'initialisation. L'inconvénient majeur de cette méthode est sa complexité. De plus, elle est sensible aux bruits et aux données aberrantes. Dans cette expérimentation, nous faisons un découpage de l'arbre hiérarchique pour que nous puissions avoir 4 *clusters*. La mesure de dissimilarité inter-classe utilisée est le « average-linkage »
- Méthode SOM (voir annexe B.4) : Cette méthode est représentative des méthodes de *clustering* basées sur des modèles. Les méthodes basées sur des modèles essaient de trouver un modèle des clusters, puis, utilisent le modèle trouvé pour classifier de nouveaux objets. La méthode SOM est basée sur un réseau de neurones et est incrémental (les vecteurs de poids peuvent être mise à jour lorsque des nouvelles données arrivent). Cette méthode est adaptée aux grandes bases de données. Par contre, le résultat dépend des valeurs d'initialisation et de la fonction de voisinage. Dans cette expérimentation, la carte auto-organisatrice est de la taille 2x2

Suivant les résultats de notre expérimentation, la méthode K-moyennes donne le meilleur résultat. Par exemple, la Figure 3.42 montre le résultat de la méthode K-moyennes sur les *strokes* extraits de la base Saint Gall. Les *strokes* dans un même *cluster* sont présentés par une même couleur.

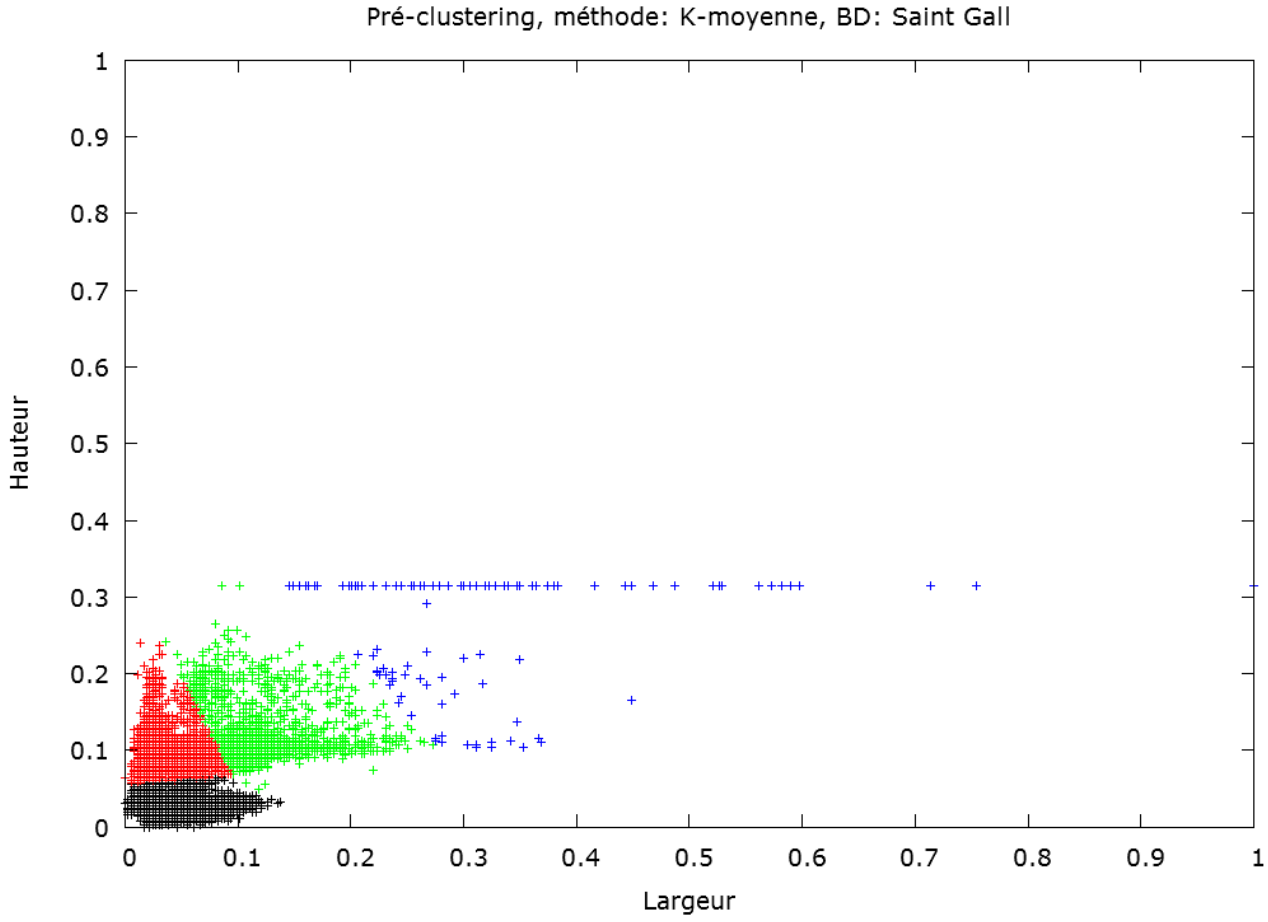


Figure 3.42 : Résultat de la méthode K-moyennes sur les strokes extraits de la base Saint Gall

La méthode DBScan regroupe bien les *strokes* de même taille, mais il est difficile de régler les paramètres de l'algorithme pour qu'il donne de bons résultats dans le cas général. Par exemple, avec les valeurs de paramètres données ci-dessus, la méthode DBScan donne de bons résultats avec la base Saint Gall, mais un mauvais résultat avec la base Parzival.

Nous concluons que la méthode K-moyenne est la méthode la plus adaptée à notre contexte. Nous utilisons donc cette méthode dans la première étape de pré-*clustering*, et nous obtenons 4 méta-clusters.

3.4.2. Deuxième étape : re-clustering

Après la première étape, les *strokes* de caractéristiques de taille similaires sont regroupés dans un même méta-*cluster*. La deuxième étape consiste en un re-*clustering* qui raffine le résultat de *clustering* de la première étape. Pour chaque *cluster* de *strokes* donné par l'étape 1, nous appliquons les méthodes de *clustering* :

K-moyennes, CAH, DBScan, SOM (voir section 3.4.1) pour les redécouper en sous-clusters.

À ce stade de l'analyse, et pour caractériser les *strokes*, nous utilisons les descripteurs suivants : l'excentricité, la rectangularité, la solidité et les boîtes de limitation (voir Annexe A). Le vecteur de caractéristiques d'un *stroke* est une concaténation des vecteurs de caractéristiques de ces descripteurs. Nous choisissons comme paramètres de formes l'excentricité, la rectangularité et la solidité parce qu'ils permettent de décrire la géométrie des *strokes* et qu'ils sont simples à calculer. Par contre, ces descripteurs sont invariants à la rotation et à l'échelle or nous ne souhaitons pas l'invariance à la rotation. Donc, les paramètres de formes sont souvent utilisés avec d'autres descripteurs pour discriminer les *strokes*. Ici, nous concaténons les vecteurs de caractéristiques des descripteurs de forme avec les vecteurs de caractéristiques des boîtes de limitation. Les « boîtes de limitation » sont des caractéristiques proposée par [Bauckhage & Tsotsos 2005]. Son idée principale est d'assimiler la forme aux rectangles de tailles différentes. Chaque rectangle couvre une région de la forme. Ce descripteur est invariant à la translation et à l'échelle. De plus, il donne des informations plus détaillées sur la forme. Il est aussi assez robuste aux bruits. La complexité pour calculer le descripteur « boîte de limitation » est peu élevée.

À ce niveau de l'analyse, il n'est pas possible de connaitre le nombre de *cluster* dans chaque méta-*cluster*. La solution envisagée consiste alors à faire varier ce nombre de clusters en définissant une fonction permettant d'identifier le meilleur nombre de *cluster*. Plus précisément, les paramètres des méthodes de *clustering* que nous modifions sont :

- Pour la méthode des K-moyennes : nous modifions le paramètre K de 2 à 100
- Pour la méthode CAH : nous modifions le nombre de clusters désiré K de 2 à 100 avec la mesure de dissimilarité inter-classe : « average linkage ».
- Pour la méthode DBScan : nous faisons varier le paramètre ϵ de 0,1 à 0,3 et le paramètre n de 2 à 8.
- Pour la méthode SOM : nous modifions la taille de la carte Kohonen de 2x2 à 10x10.

Pour choisir la meilleure solution de *clustering*, nous appliquons la méthode de consensus *clustering* que nous présentons dans la section suivante.

3.4.3. Troisième étape : consensus clustering

Nous utilisons une méthode de consensus *clustering* (voir annexe C) afin d'obtenir la solution de *clustering* ayant le plus de sens. Pour appliquer le consensus

clustering, nous utilisons la mesure NMI_{max} [Kvalseth 1987] (voir annexe C.4) pour calculer la similarité entre 2 solutions de *clustering*. Nous choisissons la mesure NMI_{max} parce qu'elle satisfait 2 propriétés : elle est métrique et normalisée. La mesure NMI_{max} ne satisfait pas la propriété « plafond constant » : la mesure de similarité moyenne des solutions de *clusterings* indépendantes devrait être une constante. Voir annexe C.4. Mais, dans notre application, le nombre d'objets (*strokes*) dans la base de données est beaucoup plus élevé que le nombre de *clusters*. La mesure NMI_{max} satisfait presque cette propriété.

Supposons que nous avons N solutions de *clustering* différentes : $C_1, C_2, \dots, C_N$. Pour chaque solution de *clustering* C_i $(i = 1..N)$, nous calculons les similarités entre C_i et les autres solutions de *clustering* $C_j (j = 1..N)$: $sim(C_i, C_j)$. La similarité $sim(C_i, C_j)$ est la mesure NMI_{max} entre 2 solutions de *clusterings* C_i et C_j : $sim(C_i, C_j) = NMI_{max}(C_i, C_j)$

Ensuite, pour chaque *clustering* $C_i (i = 1..N)$, nous calculons l'index de consensus du C_i : IC_i :

$$IC_i = \frac{1}{N} \sum_{j=1}^{N} NMI_{max}(C_i, C_j)$$

La meilleure solution de *clustering* est la solution de *clustering* dont l'index de consensus est le plus élevé.

3.4.4. Quatrième étape : extraction de prototypes

Après l'étape 3, les *strokes* sont regroupés en *clusters*. Pour chaque *cluster*, nous trouvons un *stroke* prototype qui est le plus représentatif des *strokes* dans ce *cluster*. Les invariants sont définis comme les prototypes des *clusters* trouvés dans l'étape 3.

Le prototype d'un *cluster* est le *stroke* qui est le plus proche des *strokes* dans son *cluster* et qui est le plus éloigné des *strokes* dans les autres *clusters*. Pour trouver le prototype d'un *cluster*, nous calculons, pour chaque *stroke* dans ce *cluster*, sa mesure SW (Silhouette Width) [Rousseeuw 1987]. La mesure SW pour chaque *stroke* x_i indique à quelle mesure x_i appartient à son *cluster* :

$$SW(x_i) = \frac{b(x_i) - a(x_i)}{\max\big(a(x_i), b(x_i)\big)}$$

Où :

- $a(x_i)$ est la distance moyenne entre x_i et les autres *strokes* dans son *cluster* $K(x_i)$. $a(x_i)$ représente donc l'homogénéité entre x_i et les autres *strokes* du même *cluster* :

$$a(x_i) = \frac{1}{|K(x_i)| - 1} \sum_{x_j \in K(x_i), x_j \neq x_i} distance(x_i, x_j)$$

- $b(x_i)$ est la distance entre x_i et le *stroke* d'un *cluster* $K_l \neq K(x_i)$, qui est le plus proche à x_i. $b(x_i)$ représente donc la séparation entre x_i et le *cluster* le plus proche :

$$b(x_i) = \min_{K_l \neq K(x_i)} \left(\frac{1}{|K_l|} \sum_{x_j \in K_l} distance(x_i, x_j) \right)$$

$distance(x_i, x_j)$ est la distance Euclidienne entre x_i et x_j. Nous choisissons la distance Euclidienne car cette distance est la plus classique.

Le prototype d'un *cluster* est le *stroke* dont la mesure SW est la plus élevée. Les invariants sont les prototypes des *clusters* de *strokes*.

Conclusion :

Dans cette section, nous avons présenté notre méthode d'extraction d'invariants par *clustering* de *strokes*. Lorsque les invariants sont extraits, il est nécessaire de présenter les invariants à l'utilisateur. Dans la section suivante, nous détaillons l'interface que nous avons conçue pour présenter les invariants à l'utilisateur, et permettre à ce dernier d'interagir avec les invariants comme détaillé au chapitre 5.

3.5. Visualisation d'invariants

Nous avons conçu une interface homme-machine qui permet à l'utilisateur de visualiser et d'analyser les invariants extraits automatiquement. La Figure 3.43 donne une illustration de cette interface, avec les invariants construits sur 15 pages de la base « Saint Gall ».

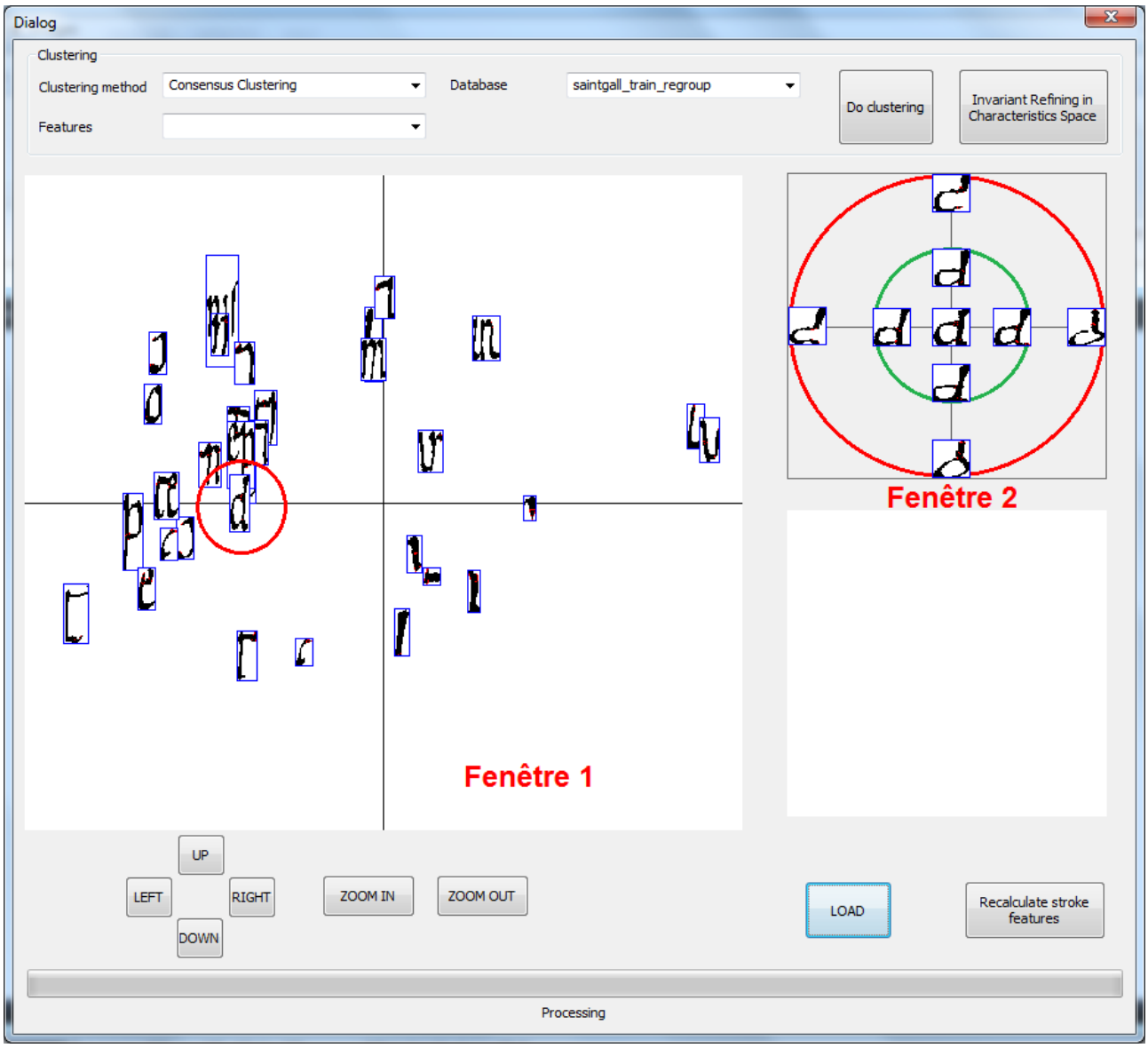


Figure 3.43 : L'interface pour visualiser les invariants

Les invariants sont présentés dans la fenêtre 1. Pour présenter les invariants dans cette fenêtre, nous appliquons une ACP (Analyse en Composantes Principales) dans l'espace de caractéristiques des *strokes* et projetons les invariants sur les 2 premiers axes principaux (avec les plus grandes valeurs propres). Les caractéristiques utilisées sont l'excentricité, la rectangularité, la solidité et les « boîte de limitation » (voir l'annexe A.3).

La projection des invariants sur un espace de 2 dimensions en appliquant l'ACP est représentative de la distribution des invariants dans l'espace des caractéristiques. C'est-à-dire que les invariants qui sont similaires se trouvent proches dans cet espace. 64% de la variance est expliquée par 2 premiers axes principaux. En présentant les invariants dans un espace de 2 dimensions, notre

système permet à l'utilisateur de juger de la qualité des invariants extraits, ainsi que des *strokes* (les *strokes* similaires sont bien groupés ? Les invariants obtenus sont bien distingués, sont suffisamment significatifs ?, *etc.*).

Quand l'utilisateur veut une information plus détaillée, il déplace la souris sur un invariant. Alors, un cercle dont le diamètre est le diamètre du *cluster* de cet invariant, est dessiné dans la fenêtre 1. Le prototype (l'invariant) est dessiné au centre de la fenêtre 2. Les *strokes* les plus proches de l'invariant sont dessinés dans le cercle intérieur, les *strokes* les plus loin de l'invariant sont dessiné dans le cercle extérieur. Cela permet à l'utilisateur d'évaluer visuellement l'homogénéité du *cluster*. Pour déterminer les *strokes* les plus proches et les *strokes* les plus éloignés de l'invariant, nous calculons la mesure SW [Rousseeuw 1987] de chaque *stroke* dans le *cluster*. L'invariant (le prototype) est le *stroke* dont la mesure SW est la plus élevée. Les *strokes* les plus proches de l'invariant ont les mesures SW les plus élevées. Les *strokes* les plus éloignés de l'invariant ont les mesures SW les moins élevées.

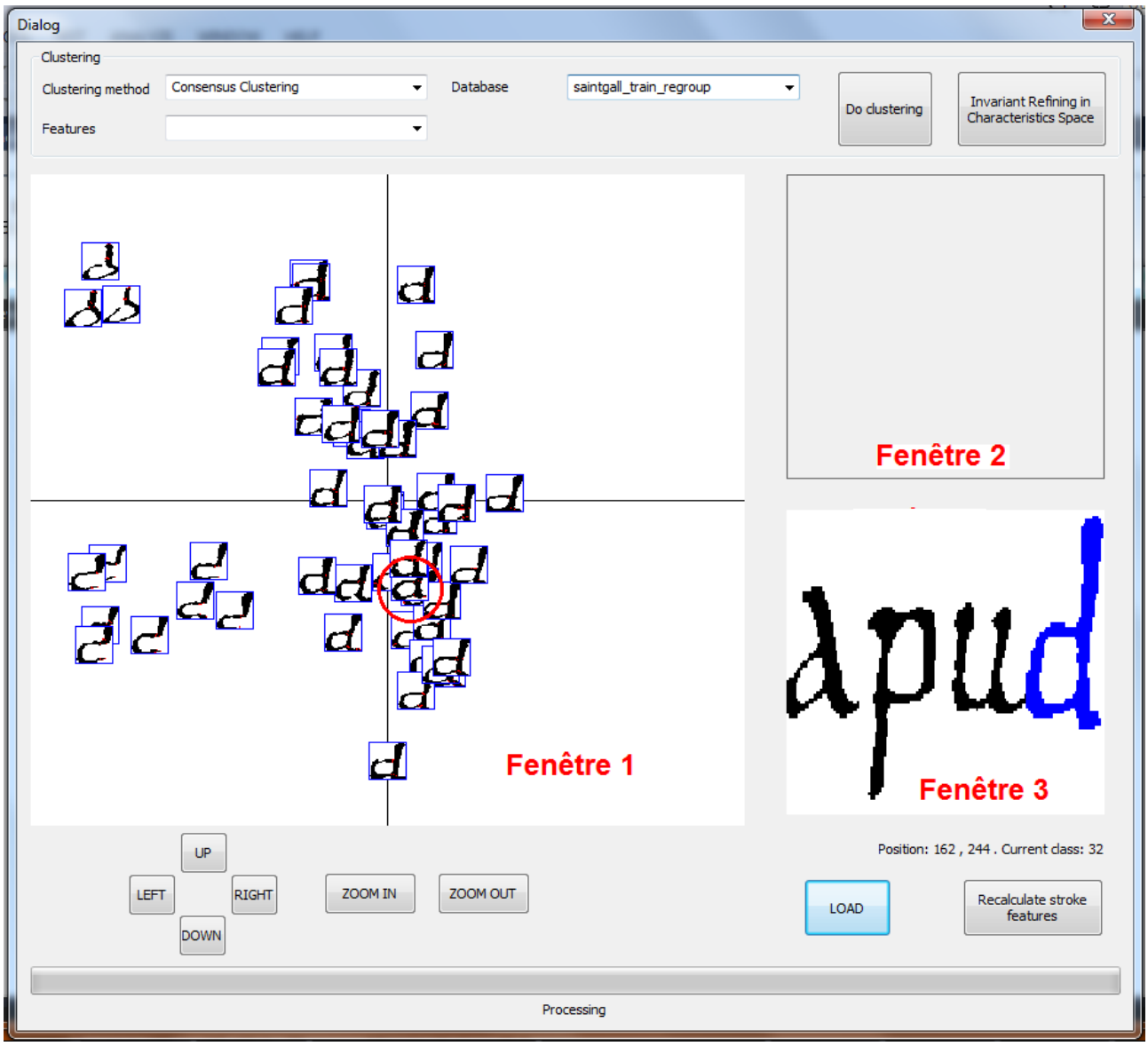


Figure 3.44 : Visualisation de tous les *strokes* dans le *cluster* d'un invariant

Pour visualiser tous les *strokes* dans le *cluster* d'un invariant, l'utilisateur peut cliquer sur cet invariant. Les *strokes* dans le *cluster* de cet invariant sont alors présentés dans la fenêtre 1. Cela permet à l'utilisateur de mieux comprendre la sémantique associée aux *strokes*, par exemple, le caractère ou le bout de caractère dans le cas de langages alphabétique. Lorsque l'utilisateur déplace la souris sur un *stroke*, la fenêtre 3 montre l'image de mot qui contient ce *stroke* et l'emplacement de ce *stroke* dans l'image (voir Figure 3.44).

3.6. Exemples du résultat de notre méthode d'extraction d'invariants

Dans cette section, nous présentons quelques exemples des invariants extraits de quelques collections de documents.

1. Exemple 1 : Invariants extraits d'une base de données synthétisée

Dans cet exemple, nous extrayons des invariants à partir d'une base de données synthétisée que nous générons. Pour générer cette base, nous prenons aléatoirement 600 mots dans un dictionnaire anglais moderne. Puis, nous utilisons la police « Arial » pour synthétiser des images de ces 600 mots. La Figure 3.45 montre un extrait de cette base.

appearing during regret adequate thereby constant mention's ban separately content's common like's causes anything rejecting bottle unacceptable grants increased classes has white considerable decade going acquired exception's permission explain expert's accuracy modern degrees seek differ carry's wise war list's largely

Figure 3.45 : Un extrait de la base de données synthétisée

La Figure 3.46 montre les invariants extraits de cette base :

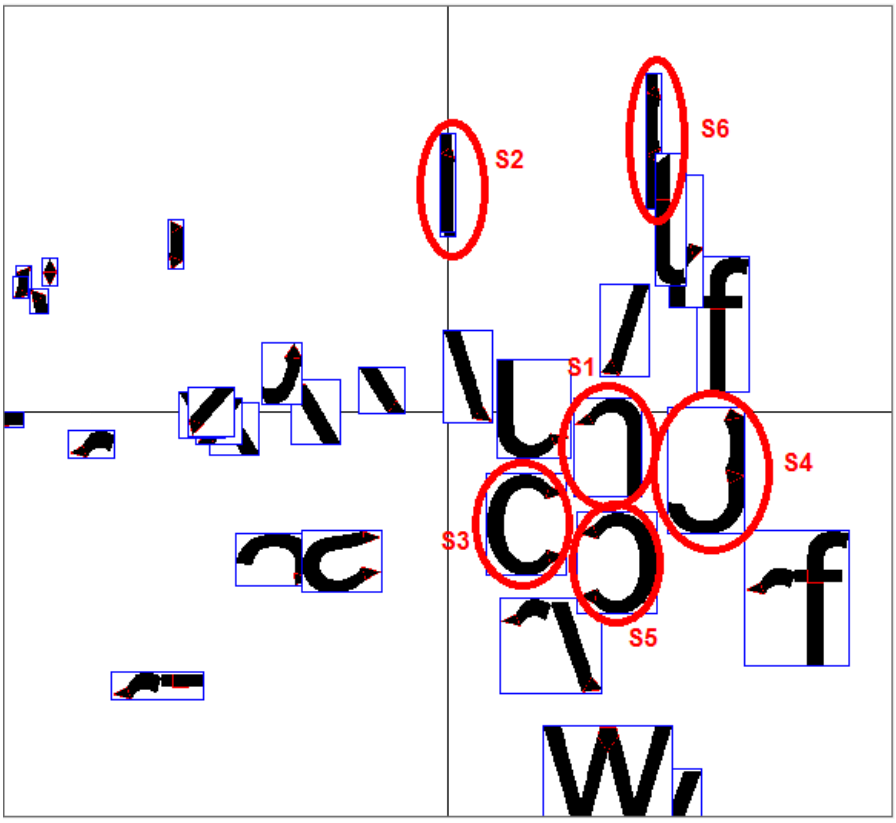


Figure 3.46 : Invariants extraits de la base de données synthétisée

A partir de cette base simple, nous arrivons à extraire complètement automatiquement des invariants qui peuvent être réutilisés par un humain pour composer des requêtes de mots. Par exemple, le tableau 3.1 montre un exemple de la composition de requêtes à partir des invariants extraits de cette base de données synthétisée.

						
				n,m		
					g	q,d
						b,p
	n,m					
		g				
		q,d	b,p			

Tableau 3.1 : Composition de requêtes par l'utilisateur à partir de quelques invariants extraits de la base de données synthétisée

2. Exemple 2 : Invariants extraits de la base « Saint Gall » :

La base « Saint Gall » contient les documents historiques manuscrits en langage Latin, écrits par un seul scripteur au 9ième siècle. Elle contient 60 pages avec 11.597 images de mots. Cette base est disponible en ligne sur http://www.iam.unibe.ch/fki/databases/iam-historical-document-database. La Figure 3.47 montre un extrait de la base « Saint Gall » et la Figure 3.48 montre son image binarisée.

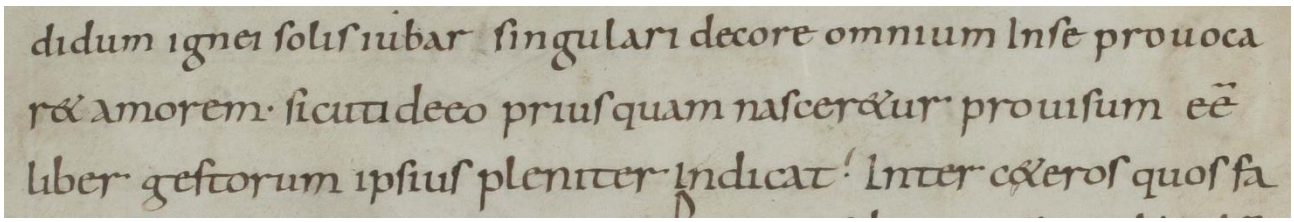


Figure 3.47 : Un extrait de la base « Saint Gall »

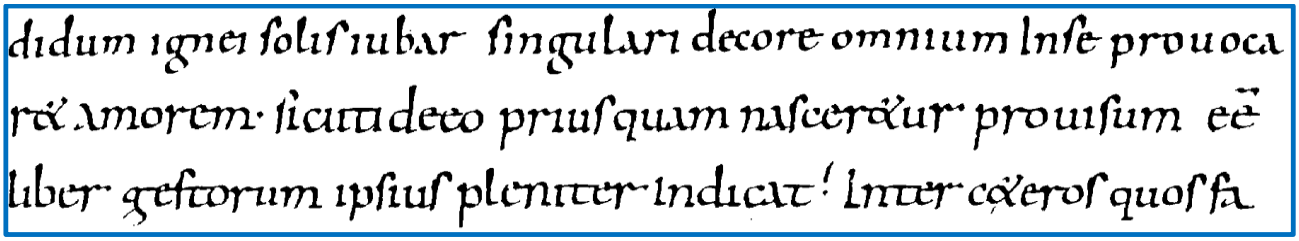


Figure 3.48 : Image binarisée d'un extrait de la base « Saint Gall »

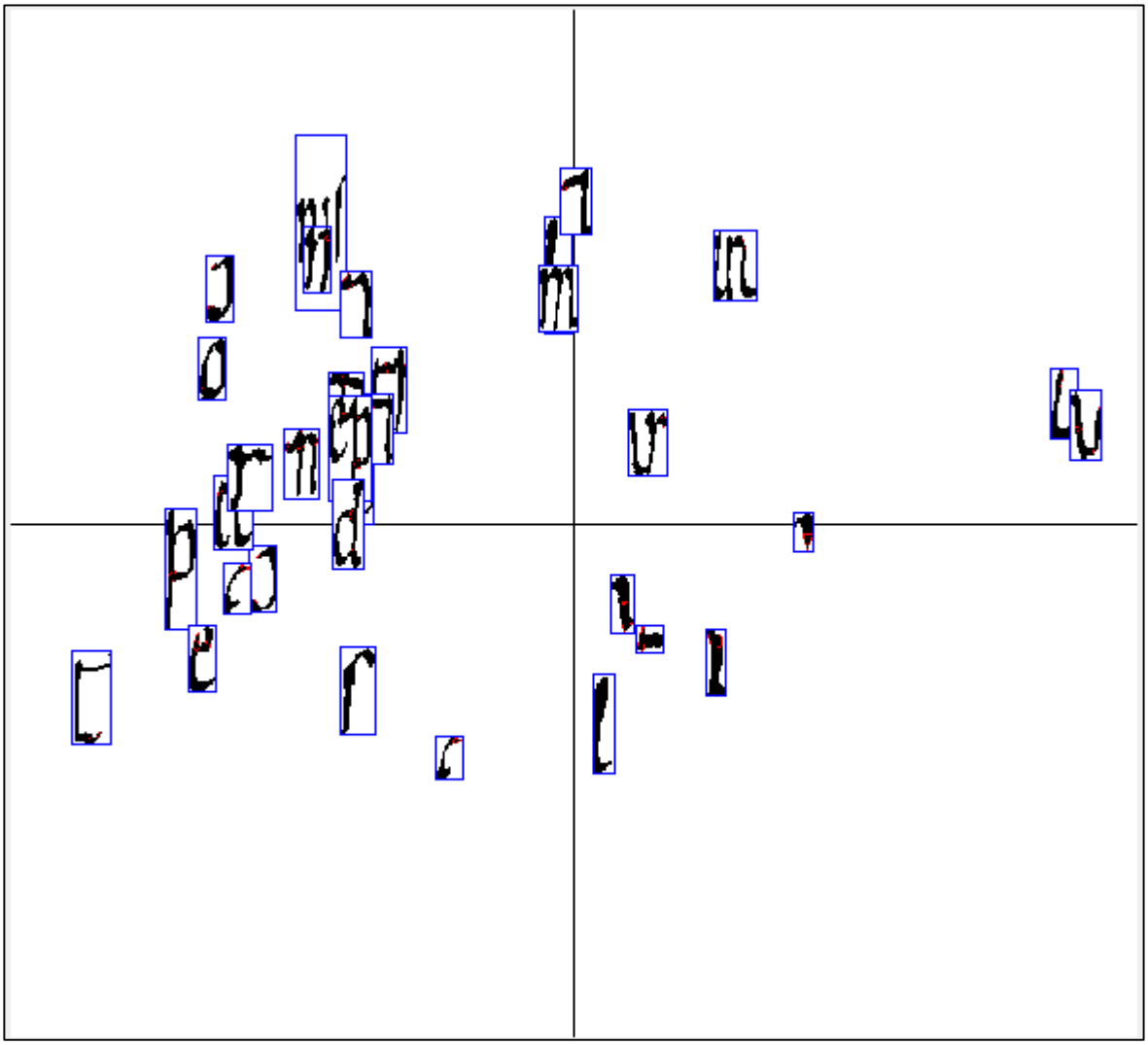


Figure 3.49 : Invariants extraits de la base « Saint Gall »

Nous prenons aléatoirement 15 pages de cette base pour extraire des invariants. Grâce à notre méthode, 31 invariants sont extraits (voir Figure 3.49). Nous arrivons à extraire quelques invariants qui peuvent être réutilisés par un humain pour composer des requêtes de. Par exemple, le tableau 3.2 montre un exemple de la composition de requêtes à partir des invariants extraits de cette base de données.

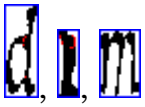
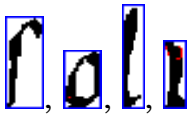
Invariants	Pour composer image de mots
	
	
	

Tableau 3.2 : Composition de requêtes par l'utilisateur à partir de quelques invariants extraits

3. Exemple 3 : Invariants extraits de la base « Washington »

La base « Washington » contient les documents historiques manuscrits du langage anglais, écrits par 2 scripteurs en 18ième siècle. Elle contient 20 pages avec 4.894 images de mots. Cette base est disponible en ligne sur http://www.iam.unibe.ch/fki/databases/iam-historical-document-database. La Figure 3.50 montre un extrait de la base « Washington » et la Figure 3.51 montre son image binarisée et normalisée (correction d'orientation, *etc.*) pour uniformiser l'apparence de l'écriture des 2 scripteurs).

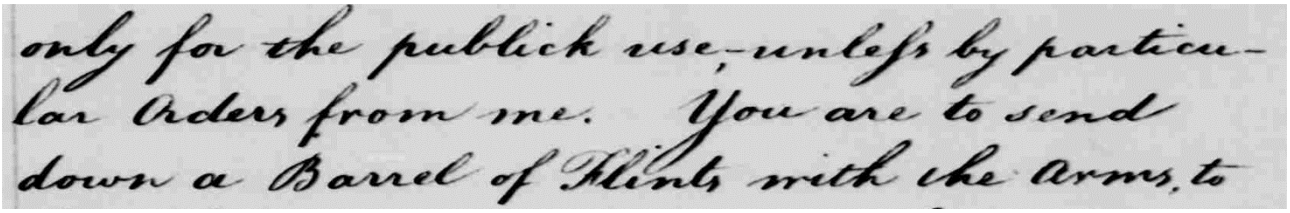

only for the publick use, unless by particu-
lar Orders from me. You are to send
down a Barrel of Flints with the Arms, to

Figure 3.50 : Extrait de la base « Washington »

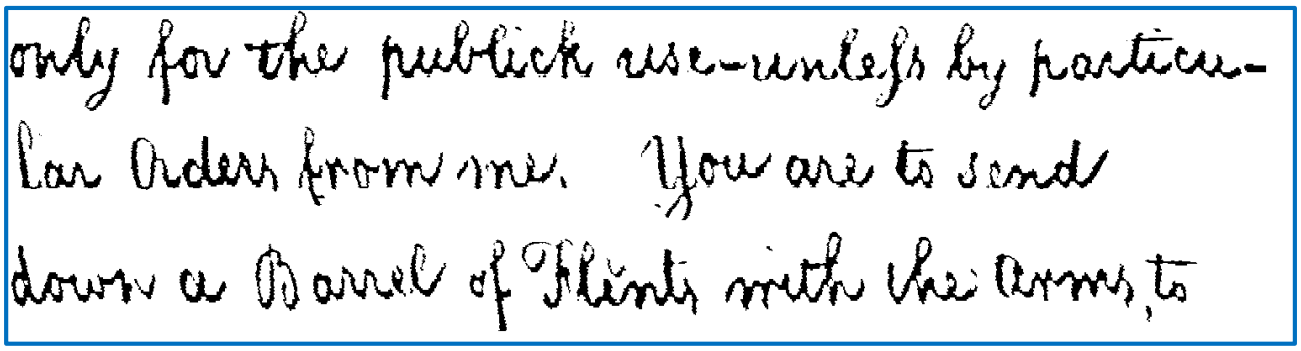

only for the publick use-unless by particu-
lar Orders from me. You are to send
down a Barrel of Flints with the Arms, to

Figure 3.51 : Image binarisée et normalisée d'un extrait de la base « Washington »

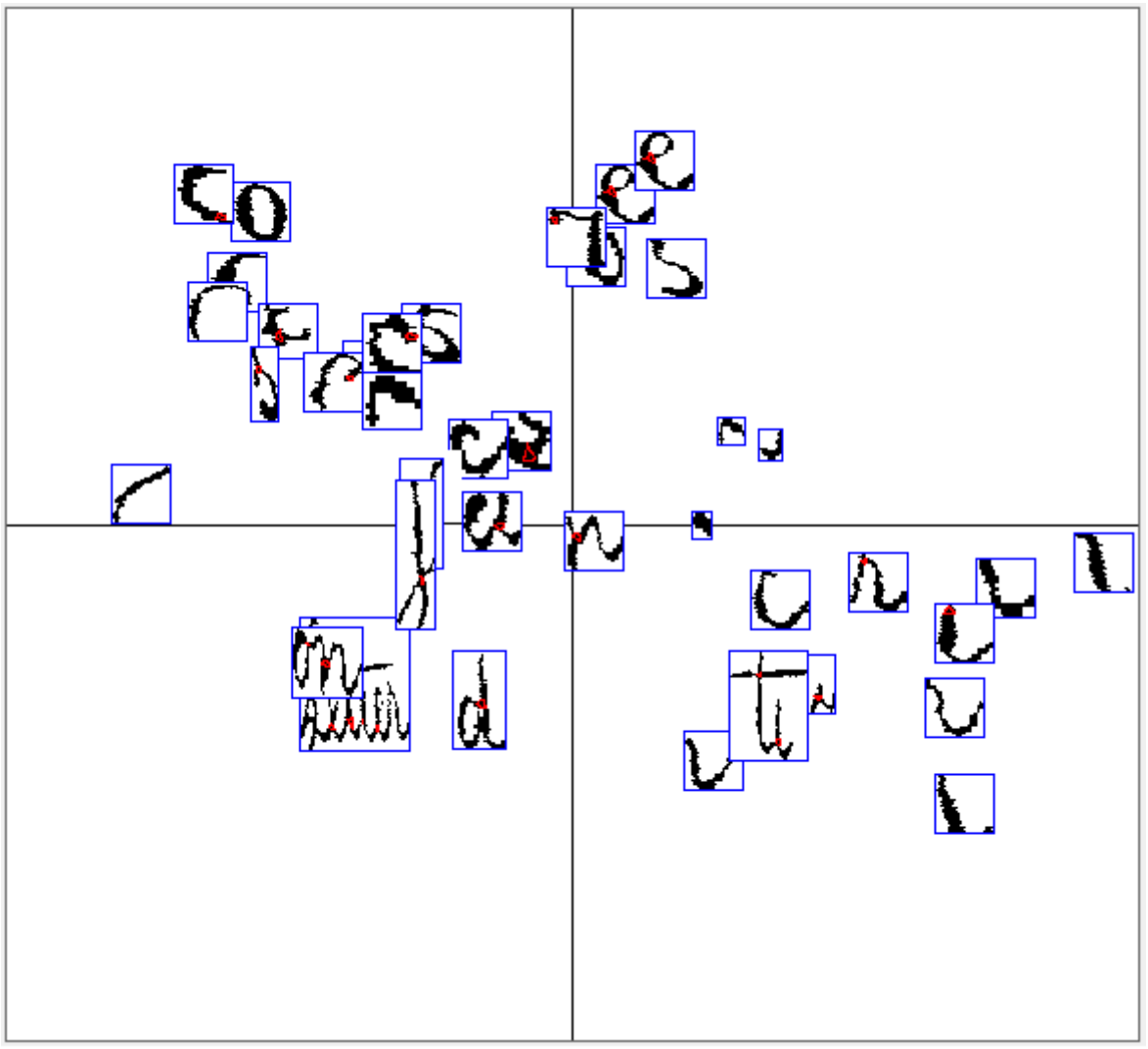


Figure 3.52 : Invariants extraits de la base « Washington »

Nous utilisons 8 pages pour extraire des invariants. Avec notre méthode, 52 invariants sont extraits (voir Figure 3.52). Nous arrivons à extraire quelques invariants qui peuvent être réutilisés par un humain pour la composition des requêtes. Par exemple, le tableau 3.3 montre un exemple de la composition de requêtes à partir des invariants extraits de cette base de données.

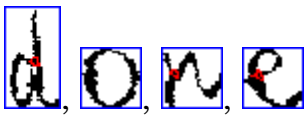
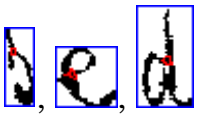
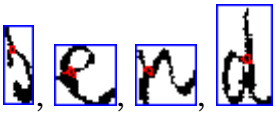
Invariants	Pour composer image de mots
	
	
	
	

Tableau 3.3 : Composition de requêtes par l'utilisateur à partir de quelques invariants extraits

Il reste encore des invariants qui nous ne semblent pas avoir un intérêt pour la composition des requêtes par un humain comme :  ,  ,  ,  , *etc.*

4. Exemple 4 : Invariants extraits d'une base synthétisée en « grec ancien »

Dans cet exemple, nous extrayons les invariants à partir d'une base que nous générons. Plus précisément, nous utilisons la police « Archaic Greek » pour synthétiser un document de 36 pages. Chaque page contient 10 lignes de mot et chaque ligne contient 10 mots. Les mots dans chaque ligne sont choisis aléatoirement dans un dictionnaire grec ancien (époque archaïque). Puis, nous appliquons des modèles de dégradation (bruit local et distorsion 3D) présentées dans [Kieu et al. 2013] pour que le document soit visuellement similaire à un document ancien. La Figure 3.53 montre un extrait de la base et la Figure 3.54 montre son image binarisée :

Figure 3.53 : Un extrait de la base synthétisée en grec ancien

Figure 3.54 : Image binarisée d'un extrait de la base synthétisée en grec ancien

Nous avons été contraints de générer cette base, car nous n'avons pas réussi à obtenir de base réelle écrite en grec ancien avec la vérité-terrain associée (mots et localisation des mots dans l'image) pour mener les évaluations dans le cadre applicatif de la recherche de mots que nous présenterons au chapitre 4.

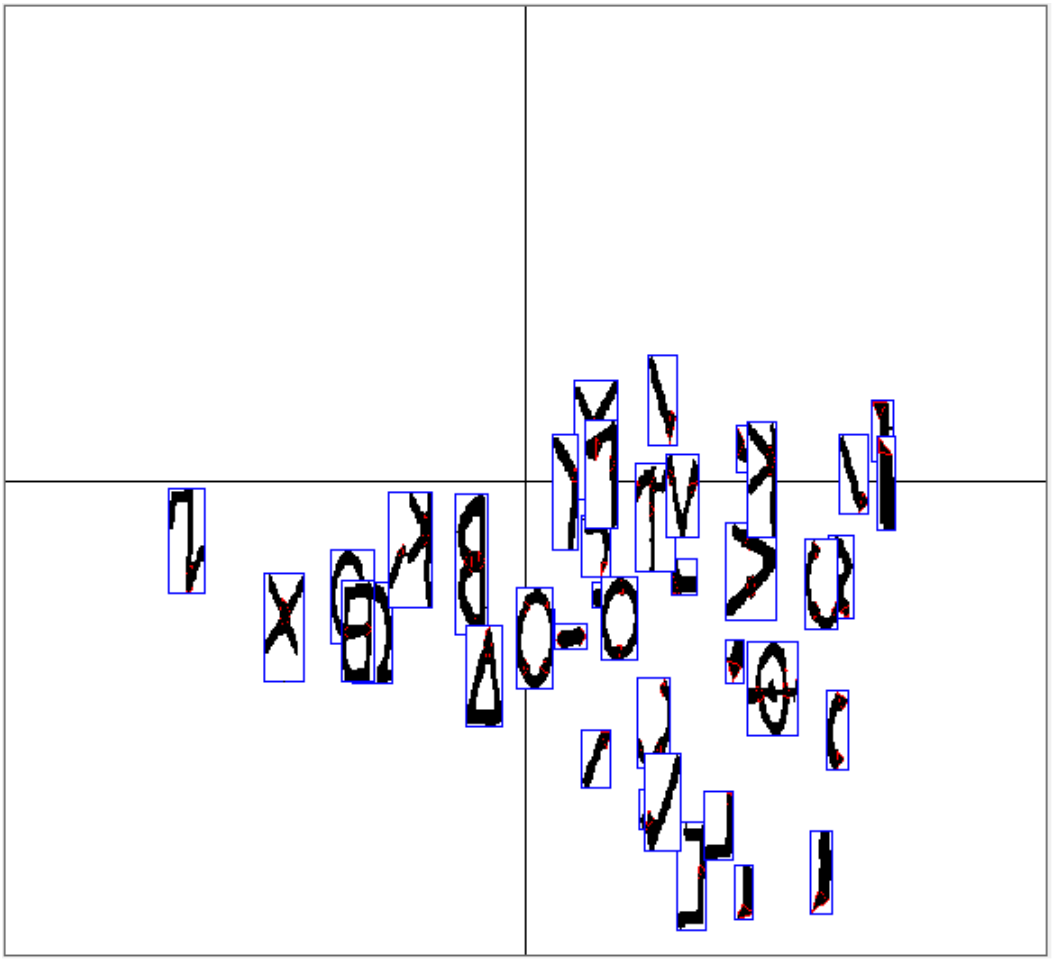


Figure 3.55 : Invariants extraits de la base « Grec ancien » synthétisée

Notre méthode permet d'extraire 43 invariants. Nous arrivons à extraire des invariants qui peuvent être réutilisés par un humain pour la composition des requêtes. Par exemple, le tableau 3.4 montre un exemple de la composition de requêtes à partir des invariants extraits de cette base de données.

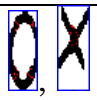
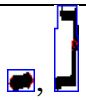
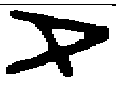
Invariants	Pour composer le caractère
	
	
	

Tableau 3.4 : Composition de requêtes par l'utilisateur à partir de quelques invariants extraits

Notamment, à cause des dégradations présentes dans le document et d'erreurs dans le regroupement de *strokes* primaires (parce que notre méthode est omni-langage), il reste encore des invariants qui nous ne semblent pas avoir un intérêt pour la composition des requêtes par un humain comme :  , *etc.*

5. *Exemple 5 : Invariants extraits de la base « Parzival »*

La base « Parzival » contient les documents historiques manuscrits du langage allemand médiéval, écrits par 3 scripteurs au 13ième siècle. Cette base contient 47 pages avec 23.478 images de mots. Cette base est disponible en ligne sur http://www.iam.unibe.ch/fki/databases/iam-historical-document-database. La Figure 3.56 montre un extrait de la base et la Figure 3.57 montre son image binarisée et normalisée.

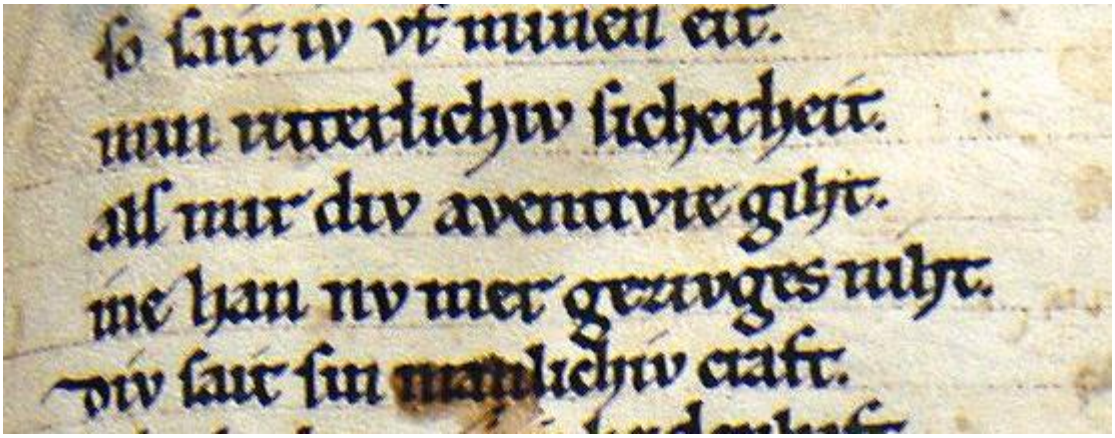


Figure 3.56 : Extrait de la base « Parzival »

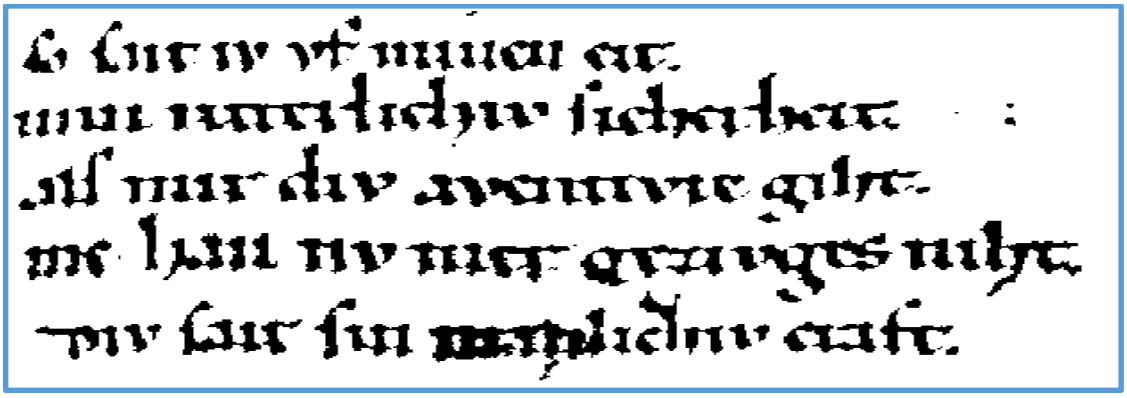


Figure 3.57 : Image binarisée et normalisée d'un extrait de la base « Parzival »

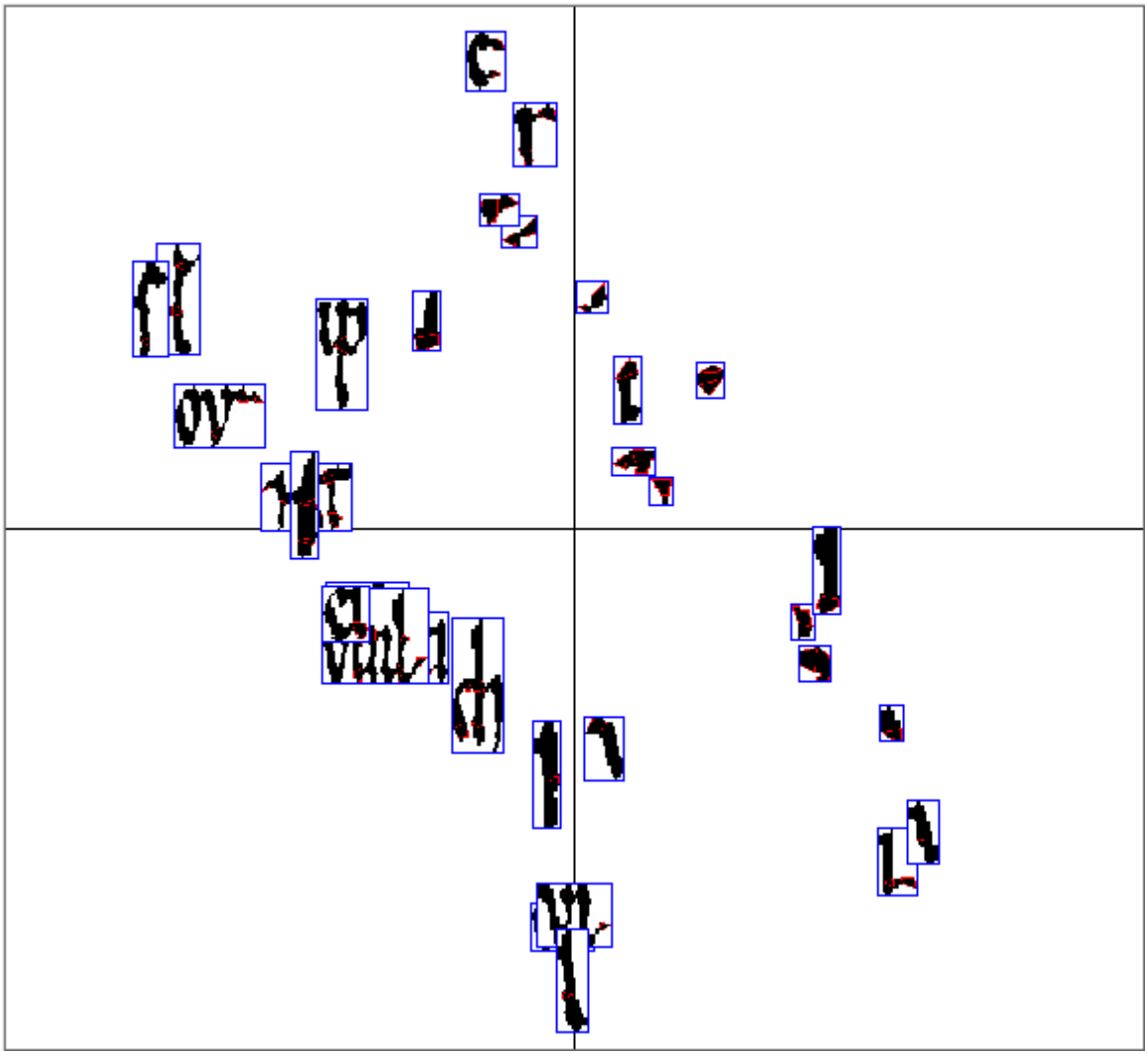


Figure 3.58 : Invariants extraits de la base « Parzival »

Nous utilisons 17 pages pour la extraire les invariants. Grâce à notre méthode, 36 invariants sont extraits (voir Figure 3.58). Nous arrivons à extraire quelques invariants qui peuvent être réutilisés par un humain pour la composition des requêtes comme :  ,  ,  ,  ,  , *etc.* Notamment à cause de la mauvaise qualité de la binarisation et de la nature cursive de l'écriture, il existe des invariants ne peuvent pas être réutilisés par un humain pour la composition des requêtes comme :  ,  ,  , *etc.*

3.7. Discussion

Nous avons présenté dans ce chapitre notre méthode d'extraction d'invariants par *clustering* de *strokes*. Nous utilisons une méthode de détection de zones ambiguës basée sur squelette pour détecter les zones ambiguës et extraire les *strokes* primaires. Pour regrouper les *strokes* primaires, nous proposons une méthode d'analyse de continuité permettant de décider de la continuité de 2 *strokes* primaires joignant une zone ambiguë. Une fois les *strokes* sont extraits, nous utilisons une méthode de *clustering* en trois étapes et d'extraction de prototypes pour extraire les invariants, qui sont les prototypes des *clusters* obtenus. Nous présentons également une interface visuelle pour présenter les invariants extraits à l'utilisateur humain. Cette interface permet à l'utilisateur d'analyser la qualité et l'intérêt pratique des *clusters* de *strokes* et les invariants extraits.

Nous avons présenté dans ce chapitre quelques exemples du résultat de notre méthode d'extraction d'invariants sur quelques bases de données, de langages et d'époques différents. Le Tableau 3.5 montre un résumé des résultats de notre méthode d'extraction d'invariants appliqué sur des bases de données différentes :

	Anglais synthétisé	Grec ancien synthétisé	Saint Gall	Washington	Parzival
Caractéristiques de la base de données	+ Parfaitement homogène + Sans dégradation, déformation, ni bruit + Aspect similaire à un document moderne imprimé	+ Homogène + Présence de dégradations, déformations et bruits + Aspect similaire à un document ancien imprimé	+ Homogène + Présence des dégradations, déformations et bruits + Document réel, ancien, manuscrit + Mono-scripteur	+ Assez homogène + Présence des dégradations, déformations et bruits + Document réel, manuscrit + 2 scripteurs	+ Homogène + Présence des dégradations, déformations et bruits + Document réel, ancien, manuscrit + Mono-scripteur
Aspect pratique des invariants extraits	Très bon	Bon	Bon	Assez bon	Mauvais
Discussion	Le document est parfaitement homogène. Il n'y a pas de dégradation, ni de bruit. Cette base nous sert de référence pour montrer l'intérêt de notre approche d'extraction d'invariant dans des conditions idéales.	Le document est homogène. L'écriture n'est pas cursive. La qualité de l'image binarisée est bonne. Donc, même s'il y a des dégradations et des bruits, nous obtenons tout de même un bon résultat.	Le document est homogène. Malgré des dégradations et des bruits, la résolution de l'image originale et la qualité de l'image binarisée sont bonnes. Donc, nous obtenons un bon résultat.	La qualité de l'image binarisée est mauvaise. La variation du style d'écriture est élevée (2 scripteurs et peu de soin apporté à l'écriture). Donc, nous obtenons un résultat moins bon que sur les 3 bases précédentes.	La qualité de l'image binarisée est mauvaise. Un nombre élevé de dégradations et de bruits est présent dans le document. Le document est homogène (mono-scripteur, écriture soignée), mais l'écriture est très cursive. Dans ces conditions, nous obtenons un mauvais résultat

Tableau 3.5 : Résumé des résultats de notre méthode d'extraction d'invariants appliqué sur quelques bases de données différents

Bien sûr certains invariants obtenus ne semblent que d'un intérêt limité dans une optique de composition de requêtes. Cela peut provenir de plusieurs causes :

- Insuffisance de la méthode de *clustering* non-supervisé :
  - Variation intra-*cluster* élevée : plusieurs *strokes* d'apparences différentes sont rangées dans un même *cluster*.
  - Variation inter-*cluster* faible : des *strokes* d'apparences similaires sont séparés dans plusieurs clusters différents.
- Insuffisance de la méthode d'extraction de *strokes* :
  - Lorsque nous utilisons une fonction générale (avec des valeurs de paramètres préétablis) pour décider la continuité de 2 *strokes* primaires

joignant une zone ambiguë, le système ne donne pas de bon résultat avec tous les types de documents. Le système peut produire les *strokes* trop grands ou trop petits et ils n'ont pas toujours d'intérêt pour la composition de requête par un humain.

- Le même type de problème peut survenir lors de l'extraction de zones ambiguës : quelques zones ambiguës ne sont pas détectées ou mal détectées. Cela peut causer des erreurs dans les *strokes* extraits.

Bien sûr, afin d'améliorer l'extraction d'invariants, il est toujours possible, comme évoqué préalablement, d'ajouter les paramètres de l'extraction de *strokes* sur une base contenant des documents du même type de langage (alphabet, logogramme, *etc.*), et ce, même en l'absence d'information spécifiques sur le langage considéré. Néanmoins, vu que les invariants sont extraits dans l'optique d'un usage ultérieur par l'humain (par exemple pour la composition de requêtes), seul l'humain peut maîtriser le niveau de sémantique requis pour juger de la qualité (ou du moins de l'intérêt pratique) des invariants. Il est donc nécessaire de fournir à l'utilisateur un outil qui lui permette de raffiner le résultat de l'extraction d'invariants. S'il n'est pas satisfait du résultat de l'extraction, l'utilisateur peut faire des raffinements nécessaires comme le fusionnement de clusters, le découpage de clusters, le découpage de *strokes*, le fusionnement de *strokes*. Nous proposons une méthode de raffinement interactive (que nous présenterons dans le chapitre suivant) pour permettre à l'utilisateur de guider le système dans les modifications à apporter aux invariants.

Afin de montrer l'intérêt pratique des invariants extraits (même automatiquement, en amont des raffinements interactifs), nous construisons un système de recherche de mots en utilisant les invariants extraits pour construire une signature structurelle. Ce système de recherche de mots est également présenté dans le chapitre suivant.

Enfin, les invariants peuvent être utilisés non seulement pour la recherche, mais aussi pour la composition de requête : l'utilisateur utilise les invariants donnés par le système pour former son image de requête. Il est nécessaire d'avoir une méthode d'évaluation des invariants pour mesurer leur capacité à former les requêtes. Nous proposons donc également dans le chapitre suivant une méthode d'évaluation d'invariants pour la composition de requêtes.

Chapitre 4 : Application à la recherche de mots

Knowing is not enough; we must apply.
Willing is not enough; we must do.

Johann Wolfgang von Goethe

4.1. Introduction

Nous avons présenté notre méthode d'extraction d'invariants dans le chapitre 3. Dans ce chapitre, nous présentons leur application à la recherche de mots. Plus précisément, ce chapitre consiste 2 parties :

- ***Système de recherche de mots*** : Dans cette section, nous présentons un système de recherche de mots en utilisant des signatures structurelles construits à partir des invariants extraits.
- ***Évaluation d'invariants*** : Lorsque les invariants sont utilisés pour la composition de requête (voir section 2.4), il est nécessaire d'avoir une méthode d'évaluation pour mesurer la qualité des invariants. Nous présentons dans cette section, la méthode d'évaluation d'invariants que nous proposons.

4.2. Contribution 3 : Système de recherche de mots utilisant une signature structurelle basée sur des invariants

Nous avons présenté dans la section 2.2 l'état de l'art des systèmes de recherche de mots. Nous présentons dans cette section, le système de recherche de mots en utilisant des invariants pour construire les signatures des mots.

La Figure 4.1 montre notre système de recherche. Le système de recherche que nous proposons est composé de 2 étapes : la construction de l'image de requête et la recherche de mots basée sur l'image de requête. Dans la première partie, l'utilisateur peut utiliser les invariants extraits pour construire une image de requête (ce qui évite d'avoir à rechercher une occurrence du mot recherché dans la collection) ou bien il peut quand même fournir une image extraite à partir de la collection de documents comme requête. Dans la deuxième étape, nous

représentons les images dans la base de données en graphe pour l'indexation. Lorsque l'utilisateur fournit sa requête (image), nous représentons cette image de requête en graphe. La représentation des images en graphes est basée sur des invariants. Ensuite, nous calculons la distance entre l'image de requête et les images dans la base de données en utilisant la mesure de dissimilarité proposée. Nous faisons un tri sur les images retournées par le système en nous basant sur leur distance à l'image de requête. Nous obtenons N_θ premières images dans la liste comme pertinentes.

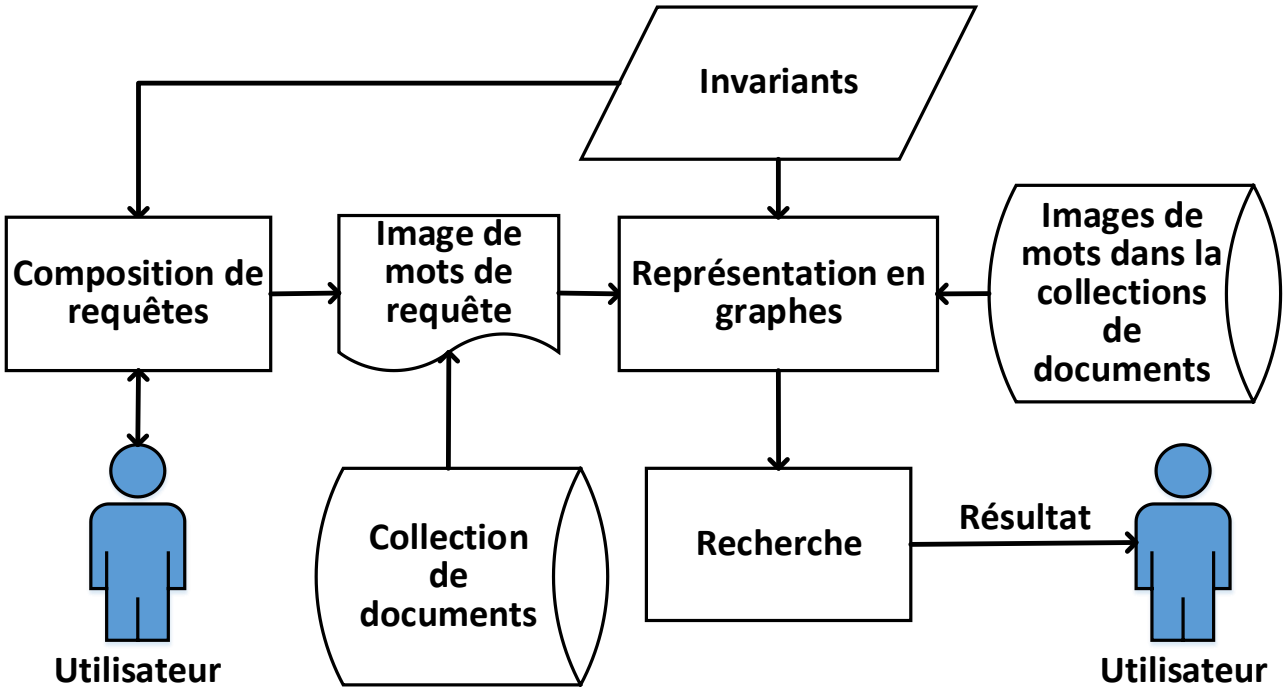


Figure 4.1 : Système de recherche de mots basé sur invariants

Nous nous concentrons ici sur la deuxième partie du système. C'est-à-dire, la recherche de mots basée sur l'image de requête. Plus précisément, cette section est composée de 3 étapes. Dans la première étape, nous introduisons la représentation en graphes des images de mots basée sur des invariants. Dans la deuxième étape, nous présentons la mesure de dissimilarité entre les images de mot, en se basant sur la représentation en graphes présentée dans la première partie. La troisième étape contient quelques expérimentations.

4.2.1. Représentation basée sur graphe

Toutes les approches dans la littérature montrent qu'il est nécessaire de disposer d'une représentation de l'image des mots doit être précise et complète. De plus, la mesure de similarité doit être robuste pour prendre en compte les variations de l'écriture. Les propriétés structurelles sont importantes pour représenter l'écriture et distinguer deux mots différents dans une même collection. La représentation basée sur des graphes est l'approche privilégiée pour capturer et modéliser les propriétés structurelles d'objets. De plus, la représentation basée sur des graphes est particulièrement pertinente dans notre contexte où les écritures dans la collection de documents sont principalement homogènes (documents anciens). Nous utilisons donc la représentation basée sur des graphes pour notre système de recherche. Plus précisément, nous utilisons des invariants pour construire le graphe représentant une image de mot. La construction d'un graphe représentant une image de mots est comme suit :

- Pour chaque image de mot dans la base de données, nous extrayons les *strokes* en utilisant notre méthode d'extraction de *strokes* (présentée dans le chapitre 3). Supposons que $I = (I_k | k = 1..N)$ sont les images de mot dans la base de données. Notons s_{lk} $(l = 1 \dots N_{I_k})$ les *strokes* du mot I_k, où N_{I_k} est le nombre de *strokes* dans l'image de mot I_k. L'image de mot I_k est représentée par un graphe $G(E,V)$ dont les nœuds correspondent aux *strokes* s_{lk} et les arêtes correspondent aux relations spatiales entre les *strokes*. Considérons maintenant une image de mot donnée et son graphe G. L'étiquette d'un nœud u_i (correspondant au *stroke* s_i) est l'invariant $x(s_i)$ qui est l'invariant représentant du *stroke* s_i (voir ci-dessous).
- L'arête entre deux nœuds u_i et $u_j \in G(E,V)$ représente la relation spatiale entre leurs *strokes* s_i et s_j correspondants :
    - Si deux *strokes* s_i et s_j sont dans une même composante connexe et qu'ils connectés à une même zone ambiguë, alors il y a une arête entre leurs nœuds correspondants u_i et u_j : $e = (u_i, u_j)$. L'étiquette de l'arête $e = (u_i, u_j)$ est la concaténation d'un caractère spécial : « * » et d'un codage de la relation spatiale entre deux *strokes* s_i et s_j (voir ci-dessous). Le caractère spécial « * » indique que s_i et s_j sont dans une même composante connexe.
    - Si 2 *strokes* s_i et s_j sont les 2 *strokes* les plus proches dans 2 composantes connexes différentes, il y a aussi une arête entre leurs nœuds correspondants u_i et u_j : $e = (u_i, u_j)$. Supposons que C_i et C_j sont respectivement deux composantes connexes de s_i et s_j. L'étiquette $e =$

(u_i, u_j) est la concaténation d'un caractère spécial : « + » et d'un codage de la relation spatiale entre C_i et C_j. Le caractère spécial « + » indique que s_i et s_j sont dans 2 composantes connexes différentes.

Pour trouver l'invariant représentatif d'un *stroke* s_l, nous calculons pour chaque invariant $x_i \in X$, la probabilité du *stroke* s_l de pouvoir être représenté correctement par l'invariant x_i : $p(x_i|s_l)$. L'invariant représentatif du *stroke* s_l est :

$$x(s_l) = \underset{x_i, i=1..M}{\operatorname{argmax}}\, p(x_i|s_l)$$

Nous calculons la probabilité $p(x_i|s_l)$ en adaptant un modèle de mélange de k = 2 Gaussiennes aux *strokes* du *cluster* de l'invariant x_i. Nous choisissons le paramètre k = 2 après une expérimentation pour comparer les performances du système de recherche utilisant le paramètre k de 2 à 10, et nous observons que le système utilisant le paramètre k = 2 donne le meilleur résultat.

Pour extraire la relation spatiale entre 2 *strokes* (ou 2 composantes connexes), nous trouvons les centres de gravité des pixels de ces 2 *strokes* (ou composantes connexes). La relation spatiale entre 2 *strokes* est la position relative de leurs centres de gravité, codée par 4 codages : 1 – haut, 2 – bas, 3 – gauche, 4 – droite (voir Figure 4.2)

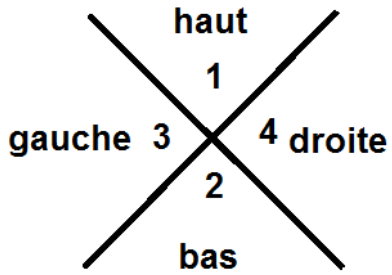
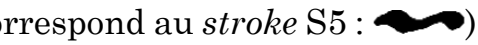


Figure 4.2 : Codage de la relation spatiale

Par exemple, La Figure 4.3 montre la représentation en graphe d'une image de mot. La Figure 4.3–a est l'image de mot. Cette image est composée en 5 *strokes* : S1 :  ; S2 :  ; S3 :  ; S4 :  et S5 :  . Le graphe qui représente cette image est composé en 5 nœuds : N1, N2, N3, N4, N5 correspondent à ces 5 *strokes* : S1, S2, S3, S4, S5 et 8 arêtes correspondent aux relations spatiales entre les *strokes*. La Figure 4.3–b montre cette représentation en graphe :

- Le nœud N1 (correspond au *stroke* S1 : ) est connecté avec le nœud N5 (correspond au *stroke* S5 : ) par 2 arêtes +1 et +2 parce que le *stroke* S1 est en bas du *stroke* S5 et ces deux *strokes* ne sont pas dans la même composante connexe.

- Le nœud N1 (correspond au *stroke* S1 : ) est connecté avec le nœud N2 (correspond au *stroke* S2 : ) par 2 arêtes +3 et + 4 parce que le *stroke* S1 est à gauche du *stroke* S2 et ces deux *strokes* ne sont pas dans la même composante connexe.

- Le nœud N2 (correspond au *stroke* S2 : ) est connecté avec le nœud N3 (correspond au *stroke* S3 : ) par 2 arêtes : *3 et *4 parce que le *stroke* S2 est à gauche du *stroke* S3 et ces deux *strokes* sont dans la même composante connexe.

- Le nœud N3 (correspond au *stroke* S3 : ) est connecté avec le nœud N4 (correspond au *stroke* S4 : ) par 2 arêtes *3 et *4 parce que le *stroke* S3 est à gauche du *stroke* S4 et ces deux *strokes* sont dans la même composante connexe.

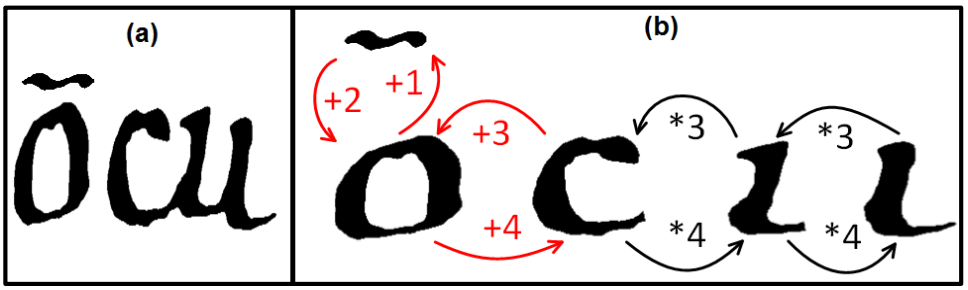


Figure 4.3 : (a) : Une image de mot. (b) Représentation en graphe de l'image de mot dans (a)

Une fois les images de mots représentées en graphes, il est nécessaire de définir une mesure de dissimilarité entre deux images de mots. Dans la section suivante, nous présentons la mesure de dissimilarité utilisée.

4.2.2. Mesure de dissimilarité entre images de mots

Dans cette section, nous définissons la mesure de dissimilarité utilisée pour comparer deux graphes. La dissimilarité entre deux images de mots est basée sur la distance d'édition des graphes correspondants. La distance d'édition entre deux graphes G_1 et G_2 est définie comme suit :

$$d(G_1, G_2) = \min_{(e_1, e_2, \ldots, e_k) \in \gamma(G_1, G_2)} \sum_{i=1}^{k} c(e_i)$$

Où $\gamma(G_1, G_2)$ est l'ensemble des chemins d'édition qui transforme G_1 vers G_2. Un chemin d'édition est une liste ordonnée des opérations d'édition : $(e_1, e_2, \ldots, e_k)$, où $c(e_i)$ est le coût de l'opération d'édition e_i. Nous définissons les opérations d'édition et leurs coûts comme suit :

- La substitution de nœuds $(u_1 \rightarrow u_2)$: c'est l'opération pour transformer un nœud $u_1 \in G_1$ en un nœud $u_2 \in G_2$. Supposons que l'étiquette de u_1 est l'invariant x_1 et l'étiquette de u_2 est l'invariant x_2. Si $x_1 \neq x_2$, alors, le coût de cette opération est : $c(u_1 \rightarrow u_2) = 1$. Si $x_1 = x_2$, alors, le coût de cette opération est : $c(u_1 \rightarrow u_2) = 0$.

- L'addition de nœuds $(null \rightarrow u_2)$: c'est l'opération pour ajouter un nœud au graphe G_1 et pour le transformer en un nœud $u_2 \in G_2$. Le coût de cette opération est : $c(null \rightarrow u_2) = 1$.

- La suppression de nœuds $(u_1 \rightarrow null)$: c'est l'opération pour supprimer un nœud du graphe G_1. Le coût de cette opération est : $c(u_1 \rightarrow null) = 1$.

- La substitution des arêtes $(e_1 \rightarrow e_2)$: c'est l'opération pour transformer une arête $e_1 \in G_1$ en une arête $e_2 \in G_2$. Supposons que a_{e_1} est l'étiquette de e_1 et a_{e_2} est l'étiquette de e_2. Si $a_{e_1} = a_{e_2}$, alors, le coût de cette opération est : $c(e_1 \rightarrow e_2) = 0$. Si $a_{e_1} \neq a_{e_2}$, alors, nous assignons différents coûts aux substitutions de position relative différente, détaillées ci-après :

  - Comme nous le savons, a_{e_1} et a_{e_2} sont des chaînes de caractères. Chaque chaîne est composée de 2 parties : un caractère spécial ('*' ou '+') et un nombre encodé (1, 2, 3 ou 4). Supposons que $n(a_{e_1})$ et $n(a_{e_2})$ sont respectivement 2 nombres encodés dans les chaînes a_{e_1}et a_{e_2}.

  - Si a_{e_1}et a_{e_2} commencent avec 2 caractères différents :

- Cas 1 : Déconnexion des composantes connexes : si $n(a_{e1}) = n(a_{e_2})$ alors le coût de cette opération est : $c(e_1 \rightarrow e_2) = 0{,}25$

- Cas 2 : Déconnexion des composantes connexes avec inversion totale des relations spatiales : si $(n(a_{e_1}) = 1\ et\ n(a_{e_2}) = 2)$, ou $(n(a_{e_1}) = 2\ et\ n(a_{e_1}) = 1)$, ou $(n(a_{e_1}) = 3\ et\ n(a_{e_2}) = 4)$, ou $(n(a_{e_1}) = 4\ et\ n(a_{e_2}) = 3)$, alors, le coût de cette opération est : $c(e_1 \rightarrow e_2) = 1$

- Cas 3 : Déconnexion des composantes connexes avec rotation des relations spatiales : le coût de cette opération est : $c(e_1 \rightarrow e_2) = 0{,}75$

o Sinon, si a_{e_1} et a_{e_2} commencent avec le même caractère :

- Cas 1 : Inversion des relations spatiales : si $(n(a_{e_1}) = 1\ et\ n(a_{e_2}) = 2)$, ou $(n(a_{e_1}) = 2\ et\ n(a_{e_1}) = 1)$, ou $(n(a_{e_1}) = 3\ \text{et}\ n(a_{e_2}) = 4)$, ou $(n(a_{e_1}) = 4\ \text{et}\ n(a_{e_2}) = 3)$, alors, le coût de cette opération est : $c(e_1 \rightarrow e_2) = 1$

- Cas 2 : Rotation des relations spatiales simples : le coût de cette opération est : $c(e_1 \rightarrow e_2) = 0{,}5$

- L'addition des arêtes $(null \rightarrow e_2)$: c'est l'opération pour ajouter une arête au graphe G_1 et pour la transformer en une arête $e_2 \in G_2$. Le coût de cette opération est : $c(null \rightarrow e_2) = 1$

- La suppression des arêtes $(e_1 \rightarrow null)$: c'est opération pour supprimer une arête $e_1 \in G_1$. Le coût de cette opération est : $c(e_1 \rightarrow null) = 1$.

Le chemin d'édition dont le coût est minimal est défini comme le chemin d'édition minimal. La distance d'édition entre le graphe G_1 et le graphe G_2 est le coût du chemin d'édition minimal. Pour trouver le chemin d'édition minimal, nous pouvons par exemple utiliser l'algorithme A* [Hart *et al.* 1968]. L'algorithme A* donne la solution optimale, mais la complexité de cet algorithme est importante (NP-Complet). Pour réduire la complexité, [Riesen & Bunke 2009] propose une méthode approximative basée sur un problème d'affectation. Cette méthode reformule approximativement la distance d'édition de graphes comme le coût d'une mise en correspondant optimale entre nœuds et arêtes de 2 graphes. Elle ne donne pas la solution optimale mais une solution sous-optimale avec une complexité moins élevée que l'algorithme A*.

Afin de comparer la méthode optimale basée sur l'algorithme A* et la méthode approximative, nous proposons une expérimentation. D'abord, nous prenons

aléatoirement 10 pages de la base Saint Gall pour extraire les invariants en utilisant notre méthode d'extraction d'invariants. Ensuite, nous représentons les images de mots dans la base Saint Gall par des graphes. Comme expliqué ci-dessus, grâce à la transcription, les images de mots sont regroupées en classes (chaque classe contient les images d'un même mot). Nous choisissons aléatoirement 18 classes (18 mots). En moyenne, chaque classe contient 11 images de mots. Pour chaque classe, nous calculons la distance d'édition moyenne de tous les graphes des images dans cette classe. Nous comparons deux méthodes : la méthode optimale [Hart *et al.* 1968] et la méthode approximative [Riesen & Bunke 2009]. La Figure 4.4 montre les distances d'édition moyennes de différentes classes. La Figure 4.5 montre le temps de calcul de la méthode optimale. La Figure 4.6 montre le temps de calcul de la méthode approximative.

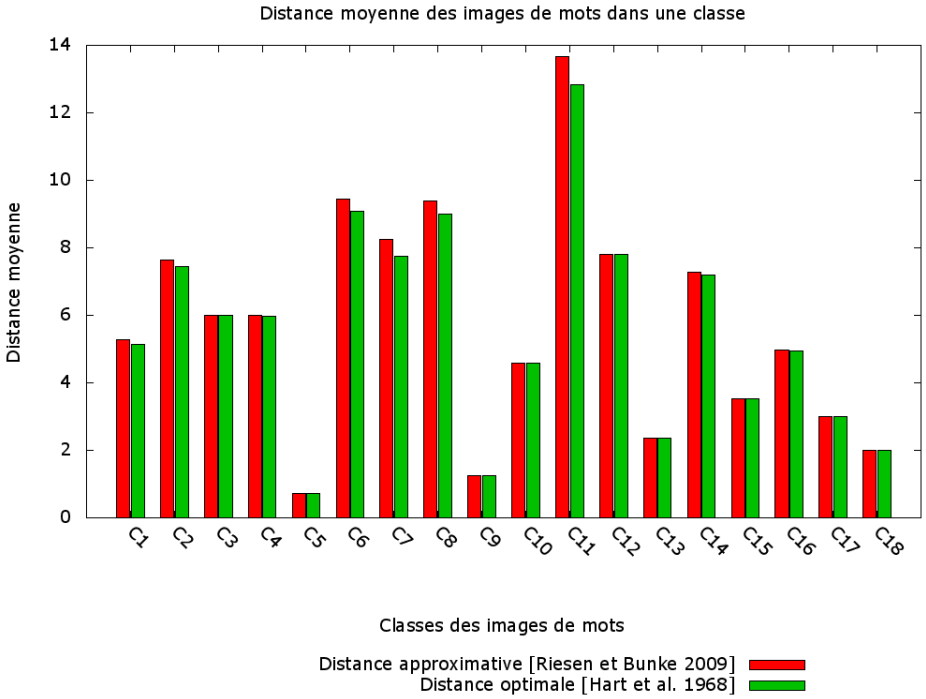


Figure 4.4 : Calcul de la distance d'édition moyenne entre graphes des images de mots dans différentes classes en utilisant la méthode de calcul optimale et la méthode de calcul approximative

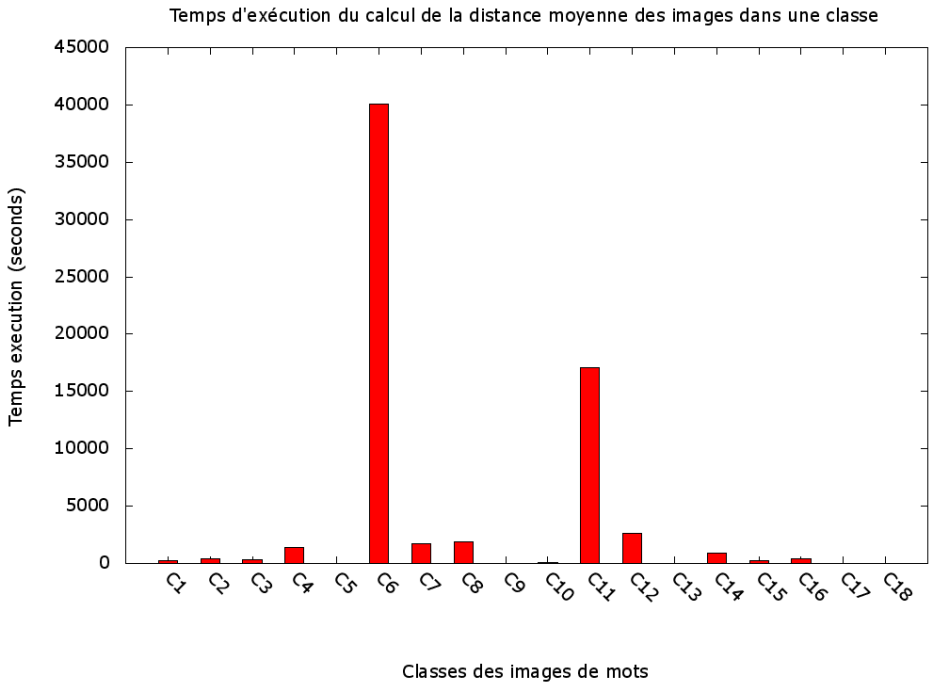


Figure 4.5 : Temps d'exécution du calcul de la distance d'édition moyenne entre graphes des images de différentes classes en utilisant l'algorithme optimal [Hart *et al.* 1968]

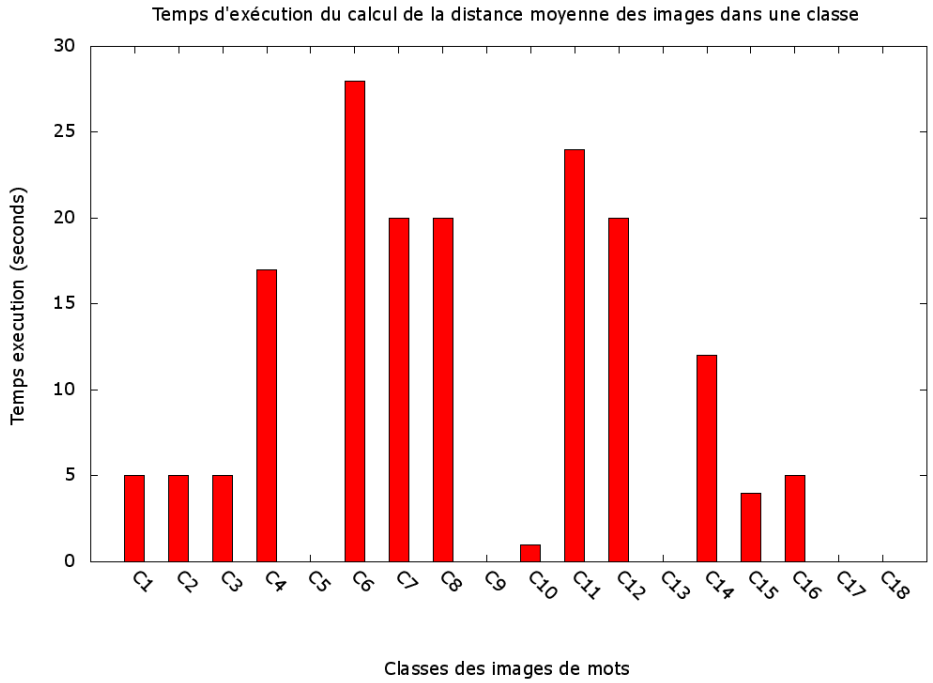


Figure 4.6 : Temps d'exécution du calcul de la distance d'édition moyenne entre graphes des images de différentes classes en utilisant l'algorithme approximatif [Riesen & Bunke 2009]

Le résultat de notre expérimentation montre que les valeurs calculées par la méthode approximative [Riesen & Bunke 2009] sont très proches des valeurs calculées par la méthode optimale. Pour vérifier l'hypothèse postulant une égalité entre ces deux types de distance, nous faisons un test de Student homoscédastique sur les distances optimales et les distances approximatives. La valeur-p du test avec une distribution bilatérale est : $p = 0{,}9120$. En constatant que la valeur-p est élevée, nous pouvons conclure que la distance approximative et la distance optimale sont égales (de façon statistique).

En revanche, le temps de calcul de la méthode approximative est beaucoup moins élevé que le temps de calcul de la méthode optimale. Nous appliquons donc la méthode approximative pour calculer la distance d'édition de graphes.

Nous définissons la distance entre deux images de mot I_1 et I_2 : $d(I_1, I_2)$ comme une normalisation de la distance d'édition de deux graphes G_1 et G_2 correspondants :

$$d(I_1, I_2) = \frac{d(G_1, G_2)}{N_{noeuds}(G_1, G_2) + N_{arrêtes}(G_1, G_2)}$$

Où $N_{noeuds}(G_1, G_2)$ est le nombre de nœuds de G_1 et G_2. $N_{arrêtes}(G_1, G_2)$ est le nombre d'arêtes de G_1 et G_2.

Conclusion :

Dans cette section, nous avons présenté la mesure de dissimilarité utilisée qui est basée sur une distance d'édition de graphes. Il existe un algorithme optimal qui calcule la vraie distance entre deux graphes, mais la complexité de cet algorithme est importante (NP-Complet). Il existe aussi un algorithme qui calcule une distance approximative. La complexité de cet algorithme est beaucoup moins élevée que l'algorithme optimale. Nous faisons une expérimentation pour comparer ces deux algorithmes. Selon le résultat de notre expérimentation qui montre que la distance approximative n'est pas significativement différente de la distance optimale (test de Student avec 18 classes de mots et 207 images de mots), nous choisissons l'algorithme approximatif pour calculer la distance d'édition de graphe.

4.2.3. Expérimentations

Cette section consiste en 3 parties. Dans la première partie, nous réalisons une expérimentation pour évaluer la performance du système de recherche proposé sur différentes bases de données. Dans la deuxième partie, nous évaluons la performance de notre système en fonction de différents niveaux de regroupements de *strokes* pour l'extraction d'invariants. Dans la troisième partie, nous réalisons quelques expérimentations pour comparer notre système de recherche avec quelques-uns des systèmes de word-spotting existants dans la littérature.

4.2.3.1. Évaluation de la performance du système de recherche proposé sur différentes bases de données

Dans cette section, nous présentons notre expérimentation pour évaluer la performance de notre système de recherche sur différentes bases de données de caractéristiques variables (âge du document, nombre de scripteurs, *etc.*). Plus précisément, pour chaque base de données, nous divisons la base en 2 parties : la première partie (la base d'extraction d'invariants) est utilisée pour l'extraction d'invariants et la deuxième partie (la base d'évaluation) est utilisée pour évaluer la performance du système de recherche. Nous utilisons les images de mots dans la base d'extraction d'invariants pour extraire les invariants. Pour l'évaluation, les images de mots qui apparaissent au moins 2 fois dans la base d'évaluation sont sélectionnées comme requêtes. Pour chaque image de requête I, nous obtenons le résultat retourné par le système de recherche. Nous faisons un tri sur les images retournées par le système en se basant sur leur dissimilarité avec l'image de requête. Nous obtenons les N_θ images les moins dissimilaires de I comme pertinentes. Grâce à la vérité-terrain, pour chaque seuil N_θ, nous calculons la précision $P(N_\theta)$ et le rappel $R(N_\theta)$ [Frakes & Baeza-Yates 1992] :

$$P(N_\theta) = \frac{|\{images\ pertinentes\} \cap \{images\ obtenues\}|}{|\{images\ obtenues\}|}$$

$$R(N_\theta) = \frac{|\{images\ pertinentes\} \cap \{images\ obtenues\}|}{|\{images\ pertinentes\}|}$$

Supposons que nous avons n images de requête : $I_i, (i = 1..n)$. Pour chaque seuil N_θ, nous obtenons des valeurs de précision différentes et des rappels différents. Nous calculons ensuite la précision et le rappel moyens $\overline{P(N_\theta)}$ et $\overline{R(N_\theta)}$.

En modifiant le seuil N_θ de 0 à 25, nous obtenons différentes valeurs de $\overline{P(N_\theta)}$ et $\overline{R(N_\theta)}$ et nous pouvons dessiner la courbe de précision-rappel. Nous calculons

également le mAP (*mean Average Precision* [Frakes & Baeza-Yates 1992]) qui est défini comme suit :

$$mAP = \frac{1}{2}\sum_{N_\theta=0}^{25}\left(\overline{P(N_\theta)} + \overline{P(N_\theta+1)}\right) * \left(\overline{R(N_\theta+1)} - \overline{R(N_\theta)}\right)$$

La courbe de précision-rappel et le mAP sont utilisés pour évaluer la performance du système de recherche.

Dans cette expérimentation, nous utilisons les bases de données suivantes :

- ***La base « Saint Gall »*** [1] : La base « Saint Gall » contient les documents historiques manuscrits du langage Latin, écrits par un seul scripteur au 9ième siècle. Cette base contient 60 pages, 4.890 mots avec 11.597 images de mots (donc, environ 192 images de mots pour chaque page). Nous utilisons 15 pages pour la base d'extraction d'invariants et 45 pages pour la base d'évaluation.
- ***La base « Parzival »*** [2] : La base « Parzival » contient les documents historiques manuscrits en langue allemande médiévale, écrits par 3 scripteurs au 13ième siècle. Cette base contient 47 pages, 4.934 mots avec 23.478 images de mots (donc, environ 500 images de mots pour chaque page). Nous utilisons 17 pages pour la base d'extraction d'invariants et 30 pages pour la base d'évaluation.
- ***La base « Washington »*** [1] : La base « Washington » contient les documents historiques manuscrits en langue anglaise, écrits par 2 scripteurs au 18ième siècle. Cette base contient 20 pages, 1.471 mots avec 4.894 images de mots (donc, environ 245 images de mots pour chaque page). Nous utilisons 8 pages pour la base d'extraction d'invariants et 12 pages pour la base d'évaluation.
- ***La base « IAM handwriting »*** [3] : La base « IAM handwriting » contient les documents manuscrits en langue anglaise. Cette base contient 1539 formulaires écrits par 657 scripteurs au 20ième siècle. Nous prenons aléatoirement 60 formulaires, écrits par 17 scripteurs pour notre expérimentation. Notre base *« IAM handwriting »* contient donc 762 mots 4.343 images de mots (donc, environ 72 images de mots pour chaque formulaire). Nous choisissons aléatoirement 20 pages pour la base d'extraction d'invariants. Nous prenons les restes (40 pages) pour la base d'évaluation.

[1] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database
[2] http://www.iam.unibe.ch/fki/databases/iam-historical-document-database
[3] http://www.iam.unibe.ch/fki/databases/iam-handwriting-database

- ***La base « Grec synthétisé »*** : La base « Grec synthétisé » est une base synthétique que nous avons créée. Nous avons été contraints de générer cette base car nous n'avons pas réussi à obtenir de base réelle écrite en grec ancien avec la vérité-terrain associée (mots et localisation des mots dans l'image) pour mener nos évaluations. Pour créer cette base, nous prenons les mots dans le dictionnaire grec pour générer un document de 36 pages. Chaque page contient 10 lignes de mot et chaque ligne contient 10 mots. Les mots dans chaque ligne sont choisis aléatoirement dans le dictionnaire. Notre base « Grec synthétisée » contient donc 1.547 mots dans le lexique avec 3.600 images de mot. Nous appliquons un modèle de dégradation et de distorsion 3D présenté dans [Kieu et al. 2013] pour que le document soit d'apparence semblable à un document ancien. La Figure 4.7 montre une page de ce document. Nous prenons 12 pages pour la base d'extraction d'invariants et 24 pages pour la base d'évaluation.

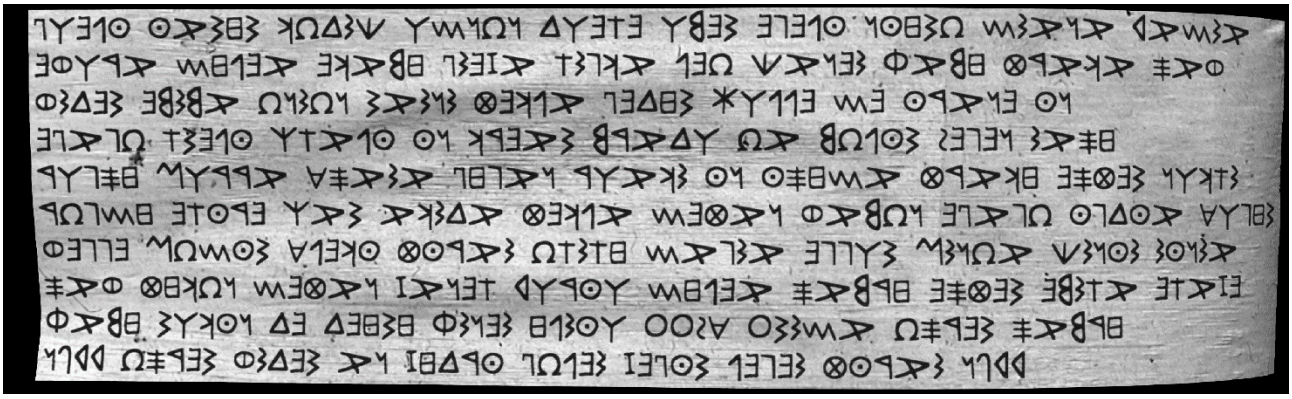


Figure 4.7 : Une page dans la base « Grec synthétisé »

La Figure 4.8 montre les courbes de précision-rappel de notre système de recherche sur les différentes bases de données détaillées plus haut :

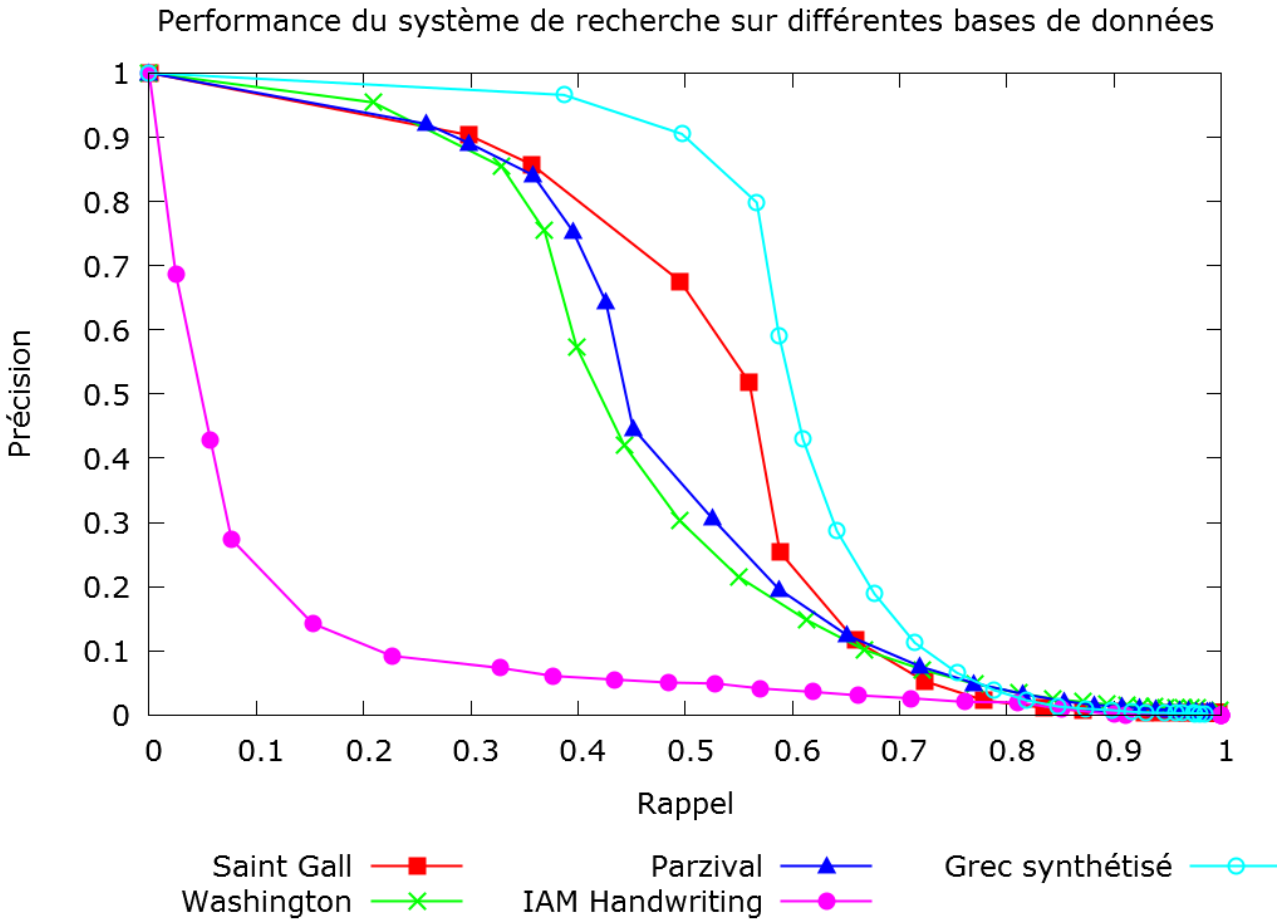


Figure 4.8 : Comparaison de la performance du système de recherche sur différentes bases de données

Le Tableau 4.1 montre les mAP de notre système de recherche sur des différentes bases de données :

Base de données	mAP
Saint Gall	0,5133
Parzival	0,4663
Washington	0,4503
IAM Handwriting	0,1989
Grec synthétisé	0,6015

Tableau 4.1 : mAP du système de recherche sur des différentes bases de données

La base « IAM Handwriting » est une base des documents non-homogènes (multi-scripteurs, les écritures sont moins soignées). Dans ce type de documents, il y a

une grande variation de l'écriture. La variation de l'écriture peut causer des erreurs dans la détection de zones ambiguës et dans le *clustering* de *strokes*. Ces variations peuvent provoquer des variations dans la représentation par graphes des images de mots (affectation à l'invariant représentant, codage des relations spatiales entre *strokes*). Pour cette raison, la performance de notre système avec la base « IAM Handwriting » ($mAP = 0{,}1989$) est la moins élevée. Nous pouvons constater qu'avec des documents plus homogènes (documents manuscrits anciens écrits par peu de scripteurs ou documents imprimés), notre système de recherche donne un meilleur résultat que les documents non-homogènes (les documents manuscrits modernes, multi-scripteurs).

Parmi les bases de documents homogènes, la performance de notre système de recherche est plus élevé sur la base « Saint Gall » que sur les bases « Parzival » et « Washington », tandis que la base « Saint Gall » contient des documents plus anciens que ceux des bases : « Parzival » et « Washington ». Cela peut être expliqué par le fait que les images binarisées de la base « Saint Gall » sont de meilleure qualité que celles des images binarisées des bases « Parzival » et « Washington ». La mauvaise qualité de la binarisation peut causer des erreurs dans la détection de zones ambiguës, dans le *clustering* de *strokes* et provoquer des variations dans la représentation par graphes des images de mots. De plus, l'écriture de la base « Saint Gall » est plus homogène que celle de la base « Washington » ; l'écriture de la base « Parzival » est plus cursive que celle de la base « Saint Gall ». Pour ces raisons, même si les documents des bases « Parzival » et « Washington » sont plus modernes que ceux de la base « Saint Gall », la performance de notre système sur la base « Saint Gall » ($mAP = 0{,}5133$) est plus élevée que les autres bases (« Parzival » ($mAP = 0{,}4663$) et « Washington » ($mAP = 0{,}4503$)). L'écriture de la base « Grec synthétisé » est la plus homogène et la moins cursive. C'est certainement la raison pour laquelle la performance de notre système sur cette base est la plus élevée ($mAP = 0{,}6015$).

Conclusion :

Nous avons présenté dans cette section une expérimentation pour évaluer la performance de notre système de recherche sur différentes bases de données de caractéristiques variables. Selon le résultat de cette expérimentation, nous pouvons conclure que notre système peut s'adapter à différents types de documents écrits avec langages alphabétiques divers sans connaissance a priori concernant le langage alphabétique utilisé. Notre système de recherche donne de meilleurs résultats avec des documents homogènes (faible nombre de scripteurs

d'écriture soignée ou impression), ce qui s'explique par la nature de notre méthode d'extraction d'invariants et de représentation structurelle des mots.

4.2.3.2. Impact de différents niveaux de regroupement de strokes primaires sur les performances du système de recherche de mots

Nous présentons cette expérimentation pour analyser l'impact de la méthode d'extraction d'invariants sur la performance du système de recherche de mots. Dans cette expérimentation, notre méthode d'évaluation est semblable à celui de la section 4.2.3.1. Mais, cette fois ci, pour une même « base d'extraction d'invariants », nous modifions les seuils de la méthode de regroupement de *strokes* primaires. En modifiant ces seuils, nous obtenons différents ensembles d'invariants. Lorsque les invariants extraits sont utilisés pour construire la signature structurelle utilisée par le système de recherche, nous comparons le système de recherche sur les ensembles d'invariants obtenus à différents niveaux de regroupement.

Plus précisément, nous modifions les seuils de la méthode de regroupement de *strokes* primaires (pour plus de détails sur les seuils, nous renvoyons le lecteur à la section 3.3.2.1) en appliquant les 3 configurations suivantes :

- Configuration 1 (regroupement de niveau 1) : Nous fixons $w_\theta = 0$ et $\gamma_\theta = 0$. En appliquant cette configuration, tous les *strokes* primaires ne sont pas regroupés.
- Configuration 2 (regroupement de niveau 2) : Nous fixons $w_\theta = 0{,}3$ et $\gamma_\theta = 0{,}15$.
- Configuration 3 (regroupement de niveau 3) : Nous fixons $w_\theta = 0{,}5$ et $\gamma_\theta = 0{,}35$. En appliquant cette configuration, les *strokes* extraits sont plus grands que les *strokes* produits en appliquant la configuration 2 car plus de *strokes* primaires sont regroupés en un seul *stroke*.

En appliquant ces 3 configurations, nous obtenons 3 ensembles d'invariants différents :

- Invariants de niveau 1 : les invariants extraits par le *clustering* des *strokes* produits en appliquant la configuration 1.
- Invariants de niveau 2 : les invariants extraits par le *clustering* des *strokes* produits en appliquant la configuration 2.
- Invariants de niveau 3 : les invariants extraits par le *clustering* des *strokes* produits en appliquant la configuration 3.

Lorsque les invariants extraits sont utilisés pour construire la signature structurelle utilisée par le système de recherche, nous avons 3 systèmes de recherche différents :

- Système de niveau 1 (S1) : le système de recherche qui utilise la signature structurelle construite par les invariants de niveau 1.
- Système de niveau 2 (S2) : le système de recherche qui utilise la signature structurelle construite par les invariants de niveau 2.
- Système de niveau 3 (S3) : le système de recherche qui utilise la signature structurelle construite par les invariants de niveau 3.

À noter qu'en section précédente (4.2.3.1), ce sont les résultats du système S2 qui ont été présentés.

Dans la suite, on parle d'invariants de « bas niveau » lorsque les seuils appliqués sont élevés avec pour résultat des invariants composés de moins de *strokes* primaires. À l'inverse, on parle d'invariants de « haut niveau » lorsque les seuils sont moins élevés.

Dans cette expérimentation, nous utilisons les 3 bases de données « Saint Gall », « Washington » et « Parzival » (pour la description plus détaillée de ces bases de données, nous renvoyons le lecteur à la section 4.2.4.1.). Sur une même base de données, nous comparons la performance des 3 systèmes de recherche présentés ci-dessus. La méthode de test est semblable à celle décrite dans la section 4.2.3.1.

La Figure 4.9 montre la comparaison des 3 systèmes de recherche sur la base « Washington » :

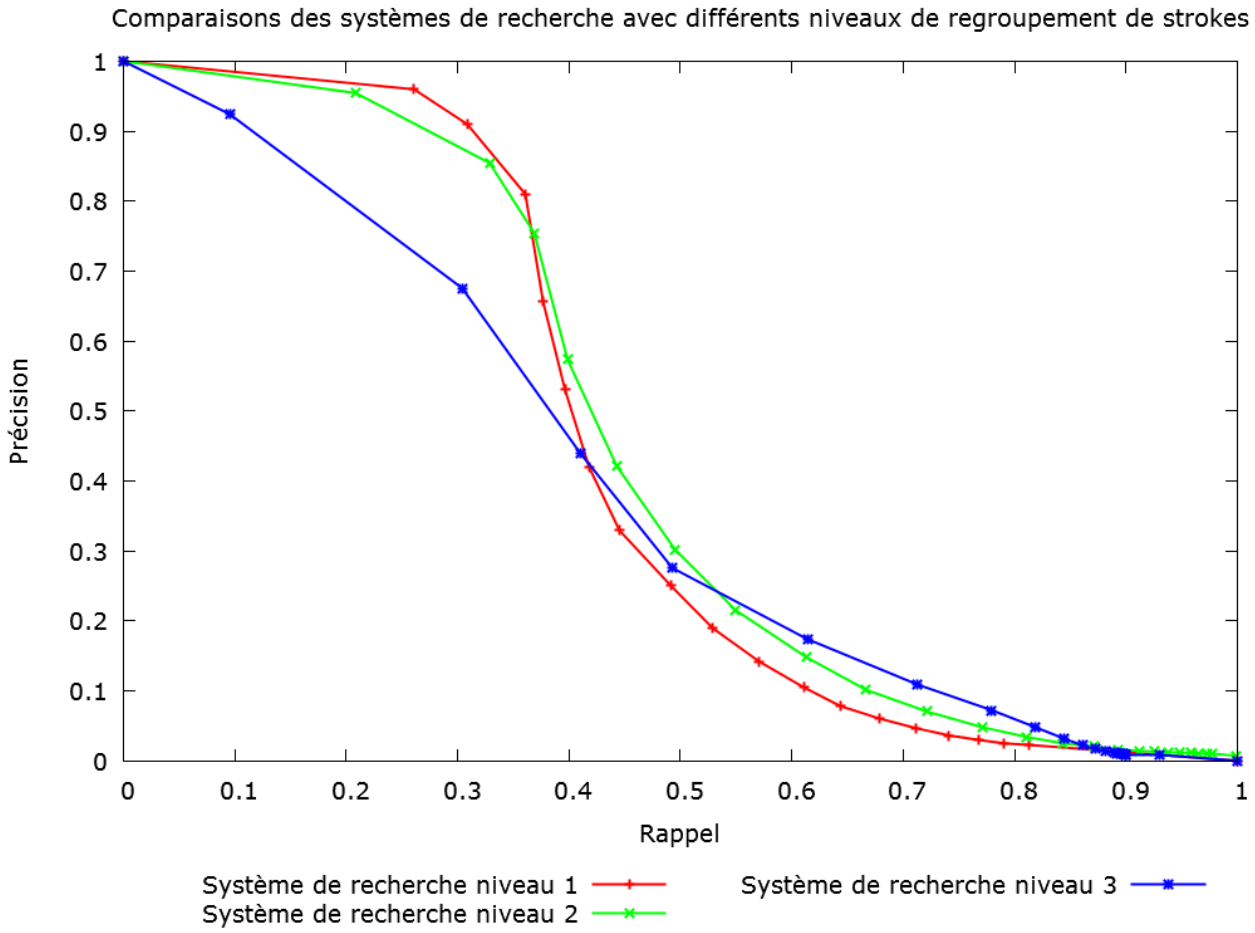


Figure 4.9 : Comparaison des systèmes de recherche sur la base Washington

Le Tableau 4.2 montre les mAP des 3 systèmes de recherche sur la base « Washington »

Systèmes de recherche	Nombre d'invariants	mAP
Système de niveau 1 (S1)	68	0,4351
Système de niveau 2 (S2)	52	0,4503
Système de niveau 3 (S3)	34	0,4000

Tableau 4.2 : mAP des systèmes de recherche sur la base Washington

Selon le résultat de notre expérimentation sur la base Washington, la performance du système S2 est la plus élevée tandis que le système S3 est la moins élevée. Cela peut être expliqué comme suit :

- La performance du système S2 est plus élevée que celle du système S1 parce que les invariants de niveau 2 sont plus « consistants » que les invariants de niveau 1. Le système S2 est donc plus robuste à des variations de l'écriture que le système S1.
- Les invariants de niveau 3 sont plus « consistants » que les invariants de niveau 2, mais la performance du système S3 est moins élevée que celle du système S2, parce que les invariants de niveau 2 sont plus spécifiques que les invariants de niveau 3. Nous observons une perte d'information lorsque nous utilisons les invariants de haut niveau pour construire des signatures structurelles.

La Figure 4.10 montre la comparaison des 3 systèmes de recherche sur la base « Parzival » :

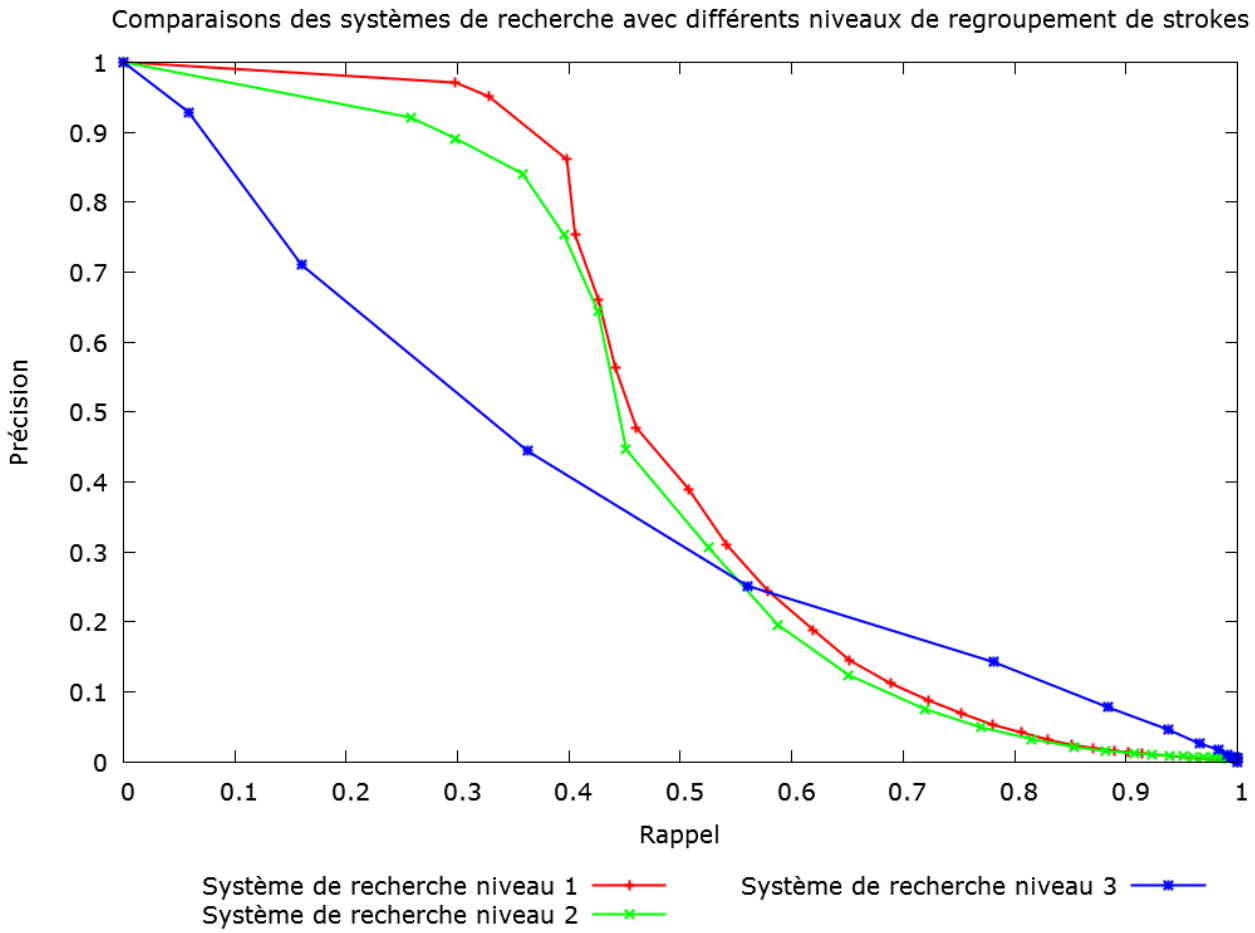


Figure 4.10 : Comparaison des systèmes de recherche sur la base Parzival

Le Tableau 4.3 montre les mAP des 3 systèmes de recherche sur la base « Parzival »

Systèmes de recherche	Nombre d'invariants	mAP
Système de niveau 1 (S1)	50	0,4992
Système de niveau 2 (S2)	36	0,4663
Système de niveau 3 (S3)	18	0,3851

Tableau 4.3 : mAP des systèmes de recherche sur la base Parzival

Selon le résultat de notre expérimentation sur la base Parzival, la performance du système S1 est la plus élevée tandis que le système S3 est la moins élevée. Cela peut être s'expliquer par le fait que l'information décrite par les invariants de niveau 1 est plus élevée que celles des autres. Les invariants de haut niveau sont plus consistants que les invariants de bas niveau, mais nous subissons une perte d'information lorsque nous utilisons les invariants de trop haut niveau pour construire des signatures structurelles.

La Figure 4.11 montre la comparaison des 3 systèmes de recherche sur la base « Saint Gall » :

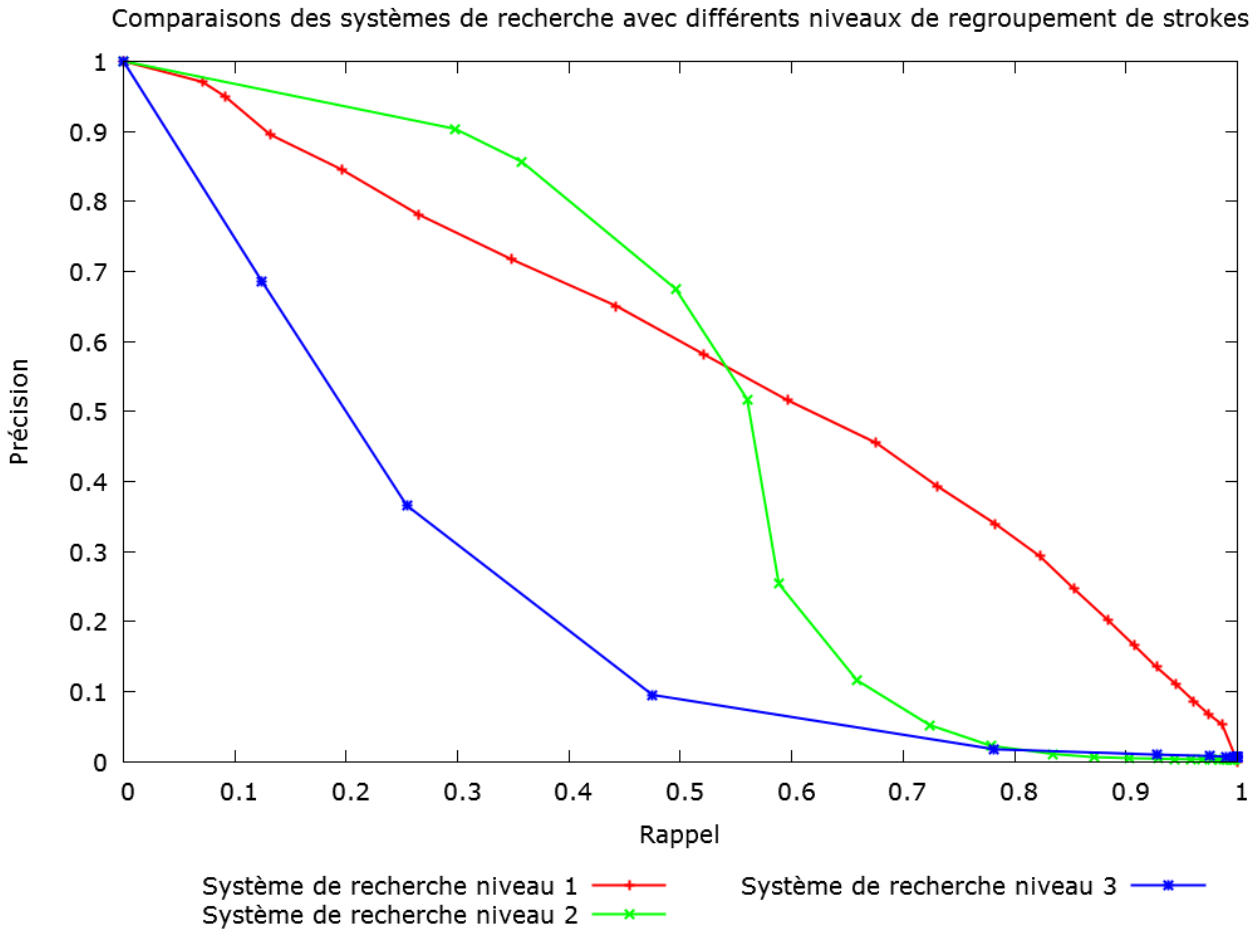


Figure 4.11 : Comparaison des systèmes de recherche sur la base Saint Gall

Le Tableau 4.4 montre les mAP des 3 systèmes de recherche sur la base « Saint Gall »

Systèmes de recherche	Nombre d'invariants	mAP
Système de niveau 1 (S1)	65	0,5772
Système de niveau 2 (S2)	31	0,5133
Système de niveau 3 (S3)	16	0,2441

Tableau 4.4 : mAP des systèmes de recherche sur la base Saint Gall

Nous observons dans ce tableau le même phénomène que sur la base Parzival

Discussion :

Dans cette section, nous avons présenté la comparaison de différents paramétrages pour le système de recherche que nous proposons en modifiant les paramètres de la méthode de regroupement de *strokes* primaires. Selon les résultats, nous pouvons conclure que les systèmes de recherche utilisant des invariants de bas niveau donnent des performances plus élevées que celles des systèmes de recherche utilisant des invariants de plus haut niveau. Cela peut expliquer par le fait que les invariants de plus bas niveau permettent de décrire plus en détail la forme des mots que les invariants de haut niveau. À l'inverse, les invariants de plus haut niveau sont plus consistants. Il nous semble préférable pour un utilisateur humain d'utiliser les invariants de haut niveau pour la composition des requêtes (voir section 2.4) plutôt que les invariants de bas niveau. Par contre, nous subissons ici une perte d'information lorsque nous utilisons des invariants de haut niveau pour construire les signatures structurelles.

4.2.3.3. Comparaison du système de recherche proposé avec les systèmes dans la littérature

Dans cette section, nous comparons la performance de notre système avec quelques-uns des systèmes de la littérature. Nous réalisons l'expérimentation sur la base de données la plus connue : la base « Washington ». Cette base est souvent utilisée pour évaluer la performance de système de word-spotting. Pour une description plus détaillée de ces bases de données, nous renvoyons le lecteur à la section 4.2.3.1.

Dans cette expérimentation, nous comparons notre système de recherche avec les systèmes dans la littérature suivants : [Manmatha *et al.* 1996], [Rusinol & Llados 2013], [Wang 2014], [Fischer *et al.* 2012], décrits dans la section 2.2.

Le Tableau 4.5 présente les descriptions des méthodes d'évaluation de chaque système différent :

Système de recherche	*Description de la méthode d'évaluation*
Système proposé	- 12 pages de la base sont sélectionnées aléatoirement pour l'évaluation. - Les images de mots qui contiennent au moins 3 caractères et apparaissent au moins 10 fois dans la base d'évaluation sont sélectionnées comme requêtes. - 1189 images de mots correspondant à 38 mots différents sont sélectionnées comme requêtes.
[Manmatha *et al.* 1996]	- La totalité des 20 pages de la base sont utilisées pour l'évaluation. - Les images de mots qui contiennent au moins 3 caractères et apparaissent au moins 10 fois dans la base d'évaluation sont sélectionnées comme requêtes. - 1847 images de mots correspondant à 68 mots différents sont sélectionnées comme requêtes. - L'évaluation est réalisée dans [Wang 2014]
[Rusinol & Llados 2013]	- La totalité des 20 pages de la base sont utilisées pour l'évaluation. - Les images de mots qui contiennent au moins 3 caractères et apparaissent au moins 10 fois dans la base d'évaluation sont sélectionnées comme requêtes. - 1847 images de mots correspondant à 68 mots différents sont sélectionnées comme requêtes. - L'évaluation est réalisée dans [Wang 2014]
[Wang 2014]	- La totalité des 20 pages de la base sont utilisées pour l'évaluation. - Les images de mots qui contiennent au moins 3 caractères et apparaissent au moins 10 fois dans la base d'évaluation sont sélectionnées comme requêtes. - 1847 images de mots correspondant à 68 mots différents sont sélectionnées comme requêtes. - L'évaluation est réalisée dans [Wang 2014]

[Fischer *et al.* 2012]	- 50% de la base de données est utilisée pour l'apprentissage. Le reste est utilisé pour la validation et le test. - Validation croisée avec 4 échantillons. Chaque échantillon contient environ 105 images de requête. - L'évaluation est réalisée dans [Fischer *et al.* 2012]

Tableau 4.5 : Descriptions des méthodes d'évaluation

La Figure 4.12 montre la courbe de précision-rappel des systèmes de recherche dans cette expérimentation.

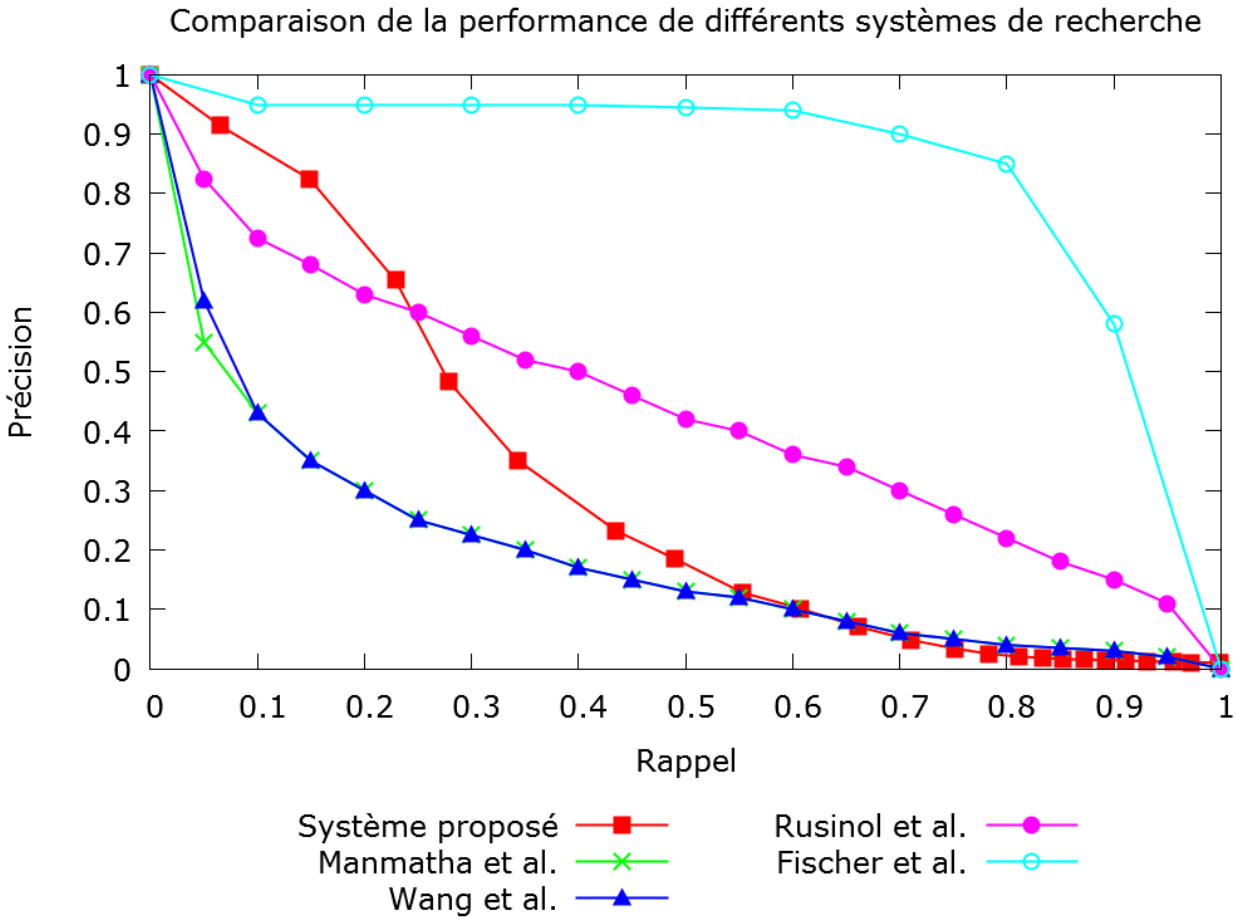


Figure 4.12 : Comparaison des différents systèmes de word spotting sur la base de données « Washington »

Le Tableau 4.6 montre les mAP des systèmes de recherche dans cette expérimentation

Systèmes de recherche	mAP
Système proposé	**0.317**
Manmatha *et al.* 1996	0.189
Wang *et al.* 2014	0.193
Rusinol & Llados 2013	0.437
Fischer *et al.*	0.851

Tableau 4.6 : mAP des différents systèmes de recherche

Selon le résultat de cette comparaison, les systèmes dont la performance est moins élevée sont le système de [Manmatha *et al.* 1996] et le système de [Wang 2014]. Ce résultat vient du fait que ces deux méthodes sont moins robustes à la variation de l'écriture que notre méthode. En particulier, notre méthode et la méthode de [Wang 2014] utilisent toutes les deux une signature structurelles pour représenter les images de mots. Un nœud du graphe représente une portion de l'image et une arête représente la relation spatiale entre deux nœuds. Une particularité de notre méthode par rapport à la méthode de [Wang 2014] est que notre méthode est basée sur l'extraction de *strokes* tandis que la méthode de [Wang 2014] est basée sur l'extraction du squelette. Or, la squelettisation est peu stable vis-à-vis de potentielles variations dans l'apparence des mots. De plus, grâce au *clustering* de *strokes* et à la représentation d'un *stroke* par un *cluster* représentatif, notre méthode est plus robuste aux variations de l'écriture que la méthode de [Wang 2014].

La performance de notre système et celui de [Rusinol & Llados 2013] sont à peu près les mêmes. En effet, tout comme notre méthode, la méthode de [Rusinol & Llados 2013] représente l'écriture d'une manière statistique (en extrayant des mots visuels issus de descripteurs SIFT), ce qui la rend plus robuste à des variations de l'écriture que celle de [Wang 2014].

Le système dont la performance est la plus élevée est le système de [Fischer *et al.* 2012]. La construction des modèles de caractères par un apprentissage supervisé et une base d'apprentissage rend cette méthode plus robuste que les autres méthodes, *vis-à-vis* de variations de l'écriture. Ainsi, en basant au niveau de caractère, la méthode de [Fischer *et al.* 2012] peut capturer l'information structurelle plus cohérente que les autres méthodes dans cette expérimentation.

4.2.4. Discussions

Nous avons présenté dans la section 4.2 le système de recherche de mots. Il est basé sur la segmentation explicite de mots et il utilise les invariants extraits pour représenter les images de mots en graphes. Nous choisissons cette représentation par graphes parce que les caractéristiques topologiques sont importantes pour représenter l'écriture, en combinaison avec des indices d'apparence (invariants). L'expérimentation présentée dans la section 4.2.3.1 montre que notre système peut s'adapter aux différents types de documents de langages alphabétiques (quel que soit le script, l'ancienneté, le type de document : manuscrit ou imprimé) sans connaissance a priori à propos du document à analyser (ni même du langage dans lequel il est écrit). Notre système de recherche donne de meilleurs résultats avec des documents d'écriture homogène qu'avec des documents d'écriture non-homogène.

Un inconvénient de notre système de recherche est qu'il ne permet pas de retrouver une portion d'un « mot » et qu'il dépend fortement de la segmentation explicite des lignes de texte en mots. C'est la raison pour laquelle dans quelques langages où les mots ne sont pas systématiquement séparés par des espaces, comme par exemple le cambodgien, notre système de recherche n'est pas adapté. Pour ce type de document, nous pouvons néanmoins utiliser les invariants dans la recherche au niveau de caractères (en cas d'écriture imprimée ou non cursive).

4.3. Contribution 4 : Méthode d'évaluation d'invariants

Nous avons présenté notre méthode d'extraction d'invariants dans le chapitre 3 avec quelques exemples dans la section 3.6. Les invariants extraits peuvent être utilisés non seulement pour la recherche mais aussi pour la composition de requêtes. Il est donc nécessaire d'évaluer la qualité des invariants extraits de manière indépendante de l'application à la recherche de mots. Dans cette section, nous présentons notre méthode d'évaluation pour évaluer des invariants.

La caractéristique la plus importante des invariants (particulièrement dans la phase de composition de requête) est la sémantique. C'est-à-dire qu'il faut vérifier si les invariants ont un intérêt pour la composition de requêtes par un humain. Mais, à cause des difficultés naturelles pour l'obtention d'une vérité terrain qui juge la pertinence des invariants extraits (le coût important de la ressource

humaine, le jugement d'une personne est subjectif, *etc.*), nous ne pouvons pas mesurer la sémantique des invariants.

Nous nous cantonnons donc à un niveau sémantique inférieur, en proposant 2 caractéristiques mesurables :

- ***Exhaustivité :*** Pour la composition de requêtes, cette caractéristique vérifie si les invariants sont suffisants pour que l'utilisateur puisse construire toutes les images de mots qu'il veut comme requêtes. L'exhaustivité élevée des invariants indique un nombre élevé des images de mots que l'utilisateur peut composer à l'aide du jeu d'invariants. En d'autres termes, cette caractéristiques vérifie si les invariants sont suffisants pour caractériser toutes les images de mots dans la base de données.
- ***Unicité :*** Cette caractéristique évalue la non redondance des invariants. Pour la composition de requêtes, les invariants dont l'unicité élevée sont en nombre peu élevé et donc a priori plus faciles à manipuler pour l'humain. Pour la recherche de mots, les invariants dont l'unicité est élevée donne une signature structurelle stable à la variation de l'écriture.

Ces 2 caractéristiques vérifient la qualité des invariants extraits. Dans les sous-sections ci-après, nous présentons plus de détails sur les mesures d'exhaustivité et d'unicité.

4.3.1. Mesure d'exhaustivité

Pour la composition de requête, l'exhaustivité indique si les invariants sont suffisants pour que l'utilisateur puisse composer toutes les images de mots qu'il veut comme requêtes. Pour la recherche de mots, cette caractéristique indique si les invariants sont suffisants pour caractériser toutes les images de mots dans la base de données. Autrement dit, la mesure d'exhaustivité indique la capacité des invariants à représenter toutes les images de mot dans la base de données. Mesurer l'exhaustivité est essentiellement nécessaire pour évaluer la qualité des invariants. Dans cette section, nous présentons la méthode d'évaluation que nous proposons pour mesurer l'exhaustivité des invariants.

L'objectif principal de notre méthode est de résoudre cette question : « Étant donnée une base d'images et un ensemble d'invariants. Quelle est la capacité de ces invariants à décrire les images dans la base ? ». Pour résoudre cette question, nous segmentons chaque image I_k dans la base en *strokes* $\{s_l, l = 1 \dots N_k\}$ (N_k est le nombre de *strokes* segmentés de l'image I_k). Pour chaque *stroke* segmenté s_l, nous

trouvons dans l'ensemble d'invariants X, l'invariant le plus similaire $x_i \in X, i = 1 \dots M$). M est le nombre d'invariants. L'image I_k est donc assimilée à une image décrite par les invariants X. La similarité $p(X|I_k)$ entre l'image originale I_k et l'image décrite par les invariants $I_k(X)$ indique la capacité des invariants X pour décrire I_k. Nous calculons, pour chaque image I_k dans la base de données, la similarité $p(X|I_k)$. La mesure d'exhaustivité est définie comme la médiane des similarités $p(X|I_k)$ de chaque image I_k dans la base de données. Pour plus de détail sur le calcul de la mesure d'exhaustivité, nous renvoyons le lecteur à la section 4.3.1.1.

Cette section est composée de 3 parties. Dans la première partie, nous présentons la méthode pour calculer la mesure d'exhaustivité que nous proposons. La deuxième partie montre une expérimentation pour valider la cohérence de la mesure d'exhaustivité que nous proposons. La troisième partie montre une expérimentation pour vérifier si nous pouvons utiliser seulement la mesure d'exhaustivité pour évaluer la qualité des invariants.

4.3.1.1. Calcul de la mesure d'exhaustivité

Supposons que $I = (I_k|k = 1..N)$ sont les images dans la base de données et $X = \{x_i|i = 1 \dots M)$ sont les invariants extraits. N est le nombre d'images dans la base de données, M est le nombre d'invariants. Supposons que l'image de mot I_k compose des *strokes* s_l ($l = 1 \dots N_k$). N_k est le nombre de *strokes* composés par l'image I_k. Pour mesurer l'exhaustivité des invariants, nous calculons, pour chaque image I_k, la similarité $p(X|I_k)$ entre l'image I_k et son image décrite par les invariants : $I_k(X)$.

Pour calculer la similarité $p(X|I_k)$ entre l'image I_k et son image décrite par les invariants : $I_k(X)$, pour chaque *stroke* s_l composé dans l'image I_k, nous calculons la probabilité du *stroke* s_l de pouvoir être représenté par l'un des invariants de X : $p(X|s_l)$. La similarité $p(X|I_k)$ est donc calculée comme suite :

$$p(X|I_k) = \prod_{l=1}^{N_{I_k}} p(X|s_l)$$

Pour calculer la probabilité du *stroke* s_l de pouvoir être représenté par l'un des invariants de X : $p(X|s_l)$, nous calculons, pour chaque invariant x_i de X, la probabilité du *stroke* s_l de pouvoir appartenir au *cluster* de l'invariant x_i : $p(x_i, s_i)$. La probabilité $p(X|s_l)$ est donc la valeur maximale des probabilités $p(x_i, s_l), x_i \in X$:

$$p(X|s_l) = \max_{i=1...M} p(x_i|s_l)$$

Pour calculer la probabilité du *stroke* s_l de pouvoir appartenir au *cluster* de l'invariant x_i, nous utilisons la même méthode présentée dans la section 4.2.1 : nous estimons la fonction de densité de la distribution des *strokes* dans le *cluster* de l'invariant x_i. Cette distribution est assimilée comme une combinaison de 2 distributions Gaussiennes. Nous appliquons l'algorithme EM [Dempster *et al.* 1977] avec k = 2 pour estimer les paramètres de cette combinaison.

La mesure de l'exhaustivité des invariants X : $E(X)$ est définie comme la médiane des probabilités $p(X|I_k), k = 1 \dots N$. Si $E(X) = 1$, toutes les images de mot dans la base de données sont parfaitement représentées par les invariants X. Si $E(X) = 0$, la plupart des images de mot dans la base de données (au moins 50%) ne peuvent pas être représentés par les invariants X.

Nous avons présenté dans cette section la méthode pour mesurer l'exhaustivité des invariants. Dans les sections suivantes, nous présenterons les expérimentations pour :

- Vérifier la cohérence de la mesure d'exhaustivité proposée.
- Vérifier si la mesure d'exhaustivité proposée seule est suffisante pour évaluer la qualité des invariants.

4.3.1.2. Expérimentation 1 : Comparaison des mesures d'exhaustivité des invariants de différents niveaux

L'objectif de cette expérimentation est de vérifier la cohérence de la mesure d'exhaustivité que nous proposons. Plus précisément, dans cette section, nous présentons une expérimentation pour comparer les mesures d'exhaustivité des invariants de différents niveaux extraits d'une même base de données. Plus précisément, pour une même base de données, nous modifions les seuils dans la méthode de regroupement de *strokes* primaires. En modifiant ces seuils, nous obtenons trois ensembles d'invariants de différents « niveaux » (voir section 4.2.3.2). Les invariants de haut niveau sont de plus grande taille (et en plus faible nombre) que ceux de bas niveau. Nous calculons, pour chaque ensemble d'invariants, sa mesure d'exhaustivité. Puis, nous faisons la comparaison des mesures d'exhaustivité différentes.

Nous réalisons notre expérimentation sur des bases de données : Base « Saint Gall », base « Washington », base « Parzival ». Nous renvoyons le lecteur à la section 3.6 pour une description plus détaillée de ces bases de données.

Le Tableau 4.7 montre la comparaison des mesures d'exhaustivité des invariants de différents niveaux extraits de la base « Saint Gall » :

	Nombre d'invariants	Mesure d'exhaustivité
Invariants de niveau 1	65	0,6731
Invariants de niveau 2	31	0,4019
Invariants de niveau 3	16	0,2452

Tableau 4.7 : Mesures d'exhaustivité des invariants de différents niveaux extraits de la base Saint Gall

Le Tableau 4.8 montre la comparaison des mesures d'exhaustivité des invariants de différents niveaux extraits de la base « Washington » :

	Nombre d'invariants	Mesure d'exhaustivité
Invariants de niveau 1	68	0,5555
Invariants de niveau 2	52	0,3750
Invariants de niveau 3	34	0,3333

Tableau 4.8 : Mesures d'exhaustivité des invariants de différents niveaux extraits de la base Washington

Le Tableau 4.9 montre la comparaison des mesures d'exhaustivité des invariants de différents niveaux extraits de la base « Parzival » :

	Nombre d'invariants	Mesure d'exhaustivité
Invariants de niveau 1	50	0,3333
Invariants de niveau 2	36	0,2915
Invariants de niveau 3	18	0,1572

Tableau 4.9 : Mesures d'exhaustivité des invariants de différents niveaux extraits de la base Parzival

Selon les résultats de notre expérimentation, la mesure d'exhaustivité des invariants de bas niveau est plus élevée que la mesure d'exhaustivité des invariants de haut niveau. Cela peut s'expliquer de la manière suivante :

- Le nombre d'images de mots qui peuvent être composées par des invariants de grande taille (des invariants de haut niveau) est intuitivement inférieur au nombre d'images de mot qui peuvent être composées par des invariants de petite taille en grand nombre (des invariants de bas niveau).
- Ces résultats expérimentaux sont cohérents avec nos conclusions de la section 4.2.3.2 pour la recherche de mots. Généralement, les systèmes de bas niveau (qui utilisent les invariants de bas niveau pour caractériser les images de mots) ont une performance plus élevée que les systèmes de haut niveau (qui utilisent les invariants de haut niveau pour caractériser les images de mots).

Nous pouvons conclure que la mesure d'exhaustivité que nous proposons est cohérente pour évaluer l'exhaustivité d'un ensemble d'invariants pour une base d'images donnée, c'est-à-dire la capacité de cet ensemble d'invariants à représenter de manière fidèle les mots de la base.

4.3.1.3. Expérimentation 2 : Comparaison des mesures d'exhaustivité des différents invariants extraits par la méthode de clustering K-moyennes avec des différents paramètres k

L'objectif de cette expérimentation est de vérifier si nous pouvons utiliser seulement la mesure d'exhaustivité pour évaluer la qualité des invariants. Nous réalisons l'expérimentation sur des bases de données suivantes : Base « Saint Gall », base « Washington », base « Parzival ». Pour chaque base de données, nous extrayons des *strokes* en utilisant la configuration de regroupement indiquée dans la section 3.3.2. Puis, nous appliquons la méthode de pré-*clustering* présentée dans la section 3.4.1 pour pré-segmenter les *strokes* en 4 *clusters*. Puis, pour chaque *cluster*, nous appliquons une méthode de *clustering* K-moyenne sur les *strokes* de ce *cluster* en modifiant le paramètre K de 2 à 50 pour générer des différents ensembles d'invariants de tailles croissantes. Ensuite, nous faisons la comparaison des mesures d'exhaustivité de ces ensembles d'invariants.

La Figure 4.13 montre la courbe des mesures d'exhaustivité des différents ensembles d'invariants extraits de la base « Saint Gall » et sa droite de régression. Elle montre aussi la mesure d'exhaustivité des invariants qui sont extraits en utilisant la méthode « consensus *clustering* » (présentée dans la section 3.4.3).

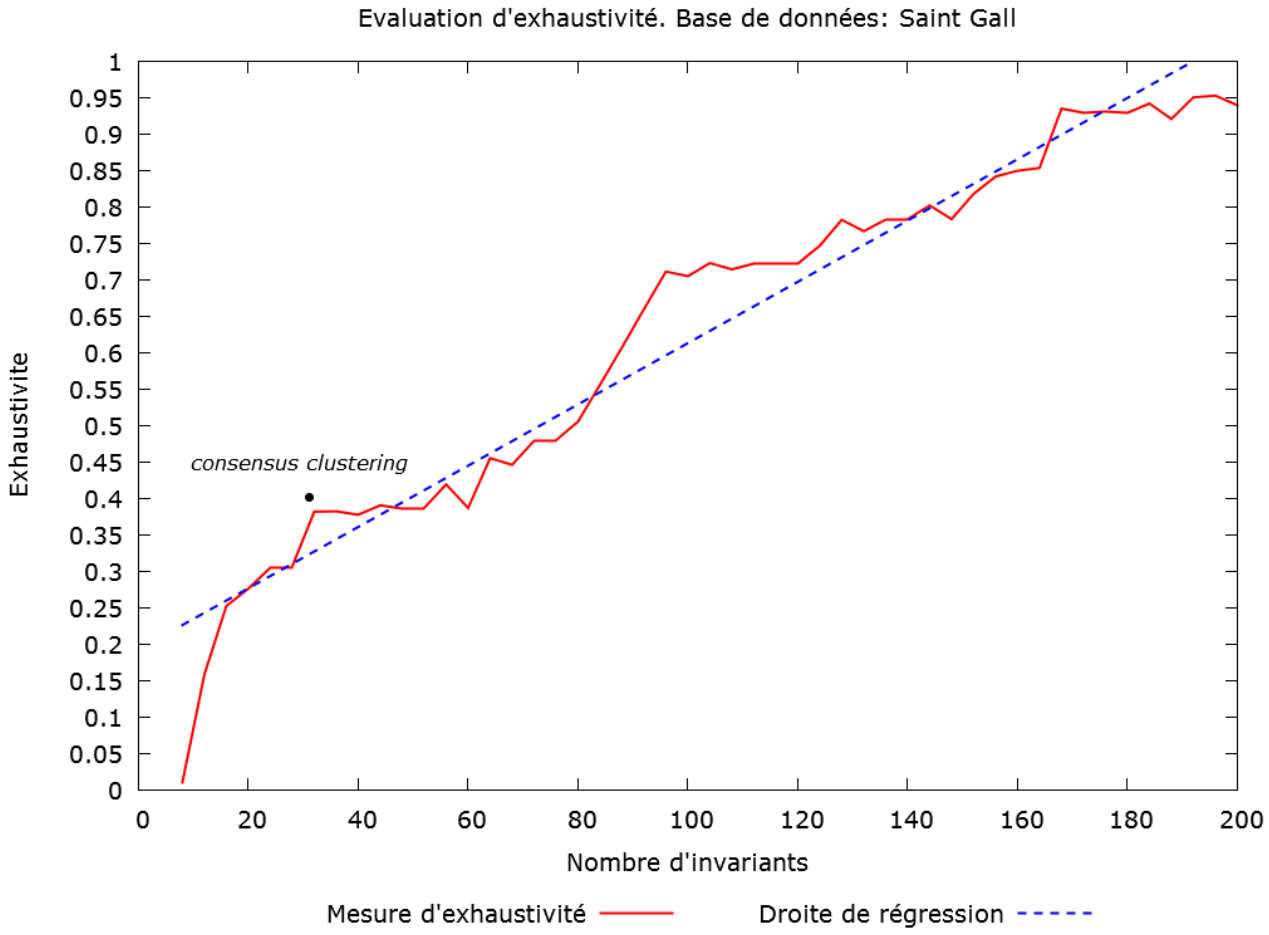


Figure 4.13 : Mesures d'exhaustivité des ensembles d'invariants extraits de la base « Saint Gall »

Nous pouvons constater que :

- Lorsque le nombre d'invariants augmente, la mesure d'exhaustivité augmente (de manière quasi-linéaire, à partir de k = 18, malgré quelques instabilités).
- Grâce à la bonne qualité des documents dans la base « Saint Gall » et à l'homogénéité élevée de l'écriture (écriture bien soignée, mono-scripteur), généralement, la mesure d'exhaustivité des invariants extraits de la base « Saint Gall » est élevée.

La Figure 4.14 montre les mesures d'exhaustivité des différents ensembles d'invariants extraits de la base « Parzival » et sa droite de régression. Elle montre aussi la mesure d'exhaustivité des invariants qui sont extraits en utilisant la méthode « consensus *clustering* » (présentée dans la section 3.4.3).

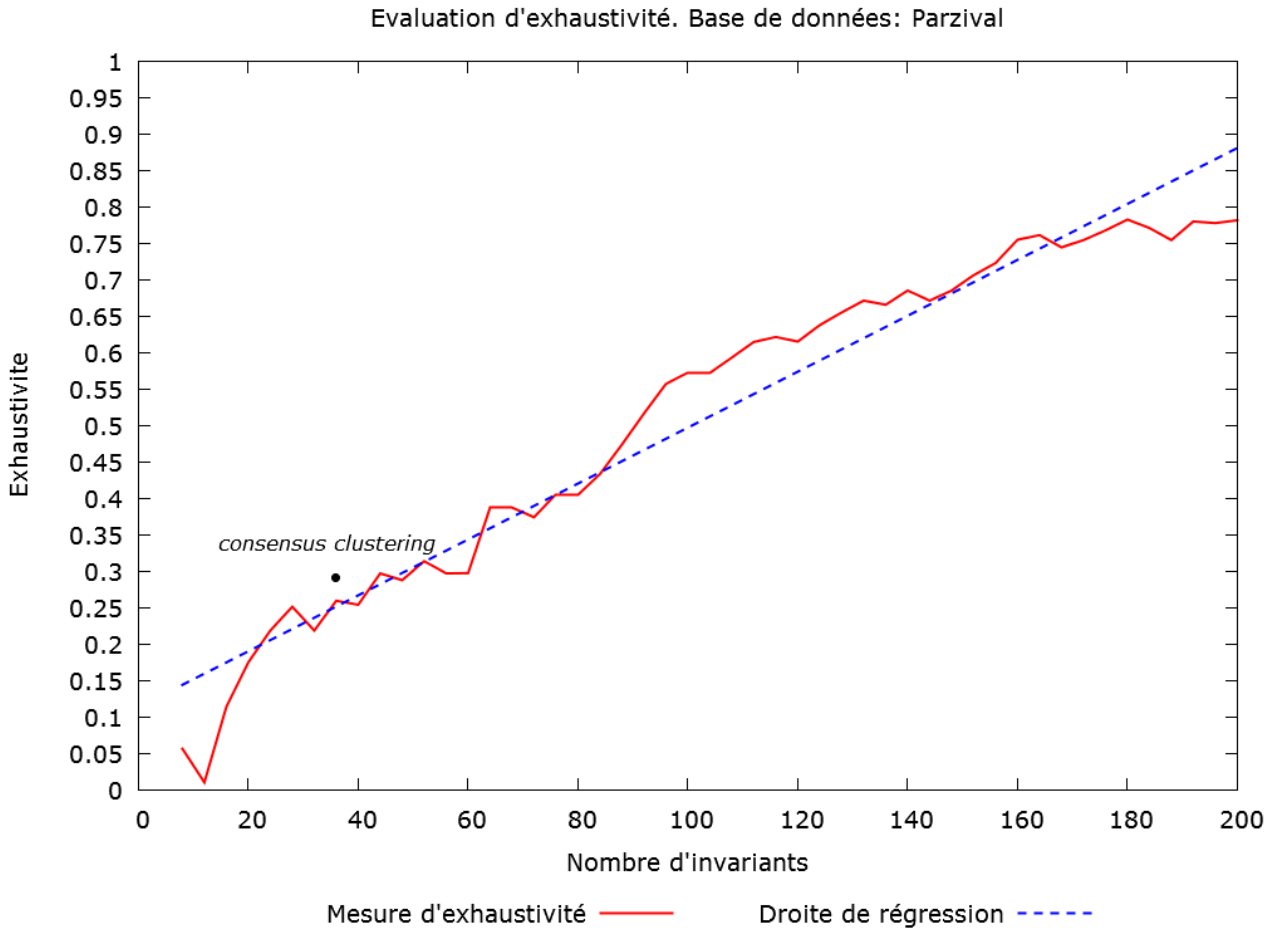


Figure 4.14 : Mesures d'exhaustivité des ensembles d'invariants extraits de la base « Parzival »

Nous pouvons constater la même tendance (lorsque le nombre d'invariants augmente, la mesure d'exhaustivité augmente) de manière quasi-linéaire à partir de k = 22. Mais, cette fois-ci, certainement à cause de la mauvaise qualité des documents de la base « Parzival », généralement, la mesure d'exhaustivité des invariants extraits de la base « Parzival » est moins élevée que celles des invariants extraits de la base Saint Gall.

La Figure 4.15 montre les mesures d'exhaustivité des différents ensembles d'invariants extraits de la base « Washington » et sa droite de régression. Elle montre aussi la mesure d'exhaustivité des invariants qui sont extraits en utilisant la méthode « consensus *clustering* » (présentée dans la section 3.4.3).

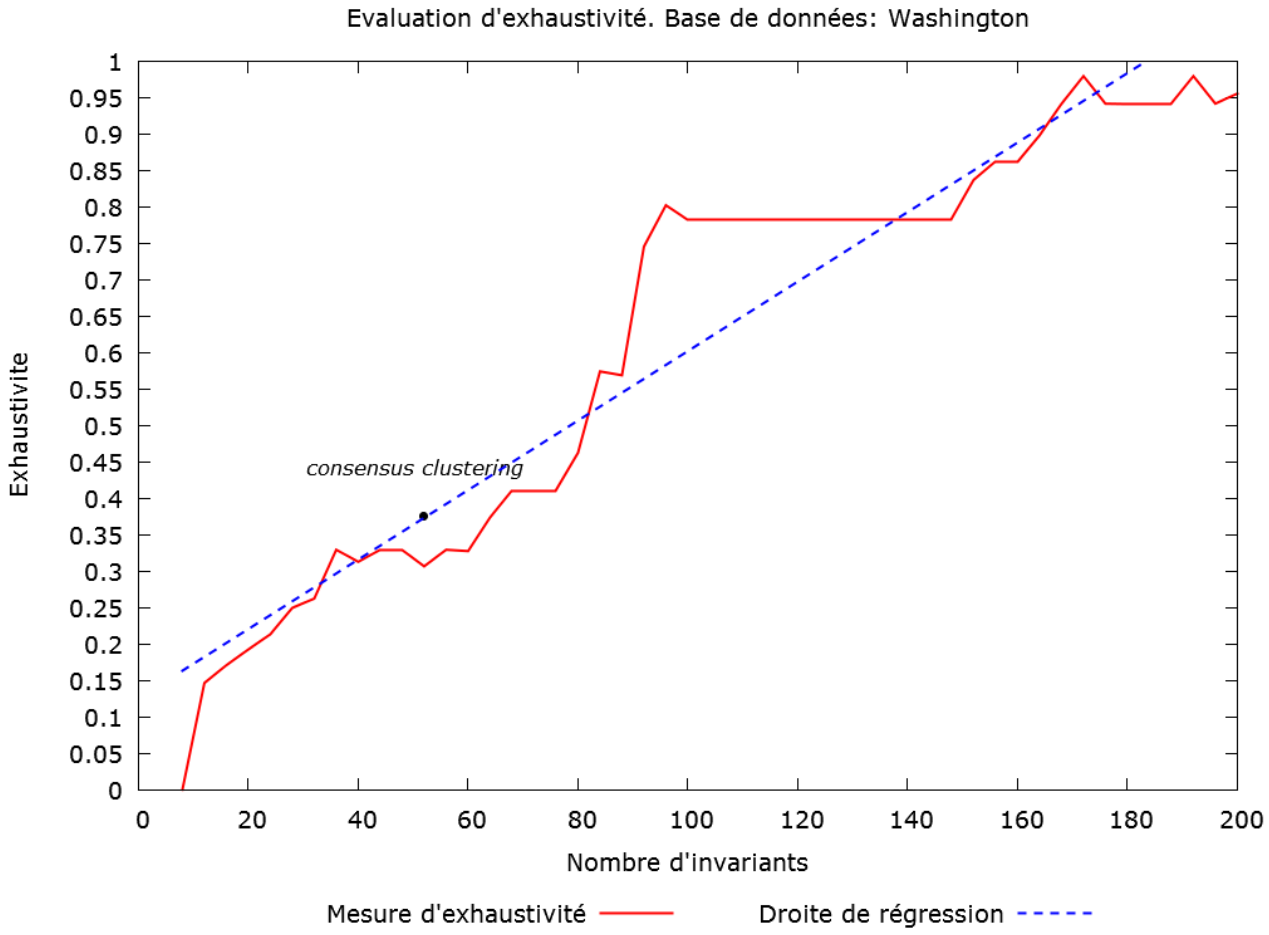


Figure 4.15 : Mesures d'exhaustivité des ensembles d'invariants extraits de la base « Washington »

Nous pouvons constater la même tendance générale (lorsque le nombre d'invariants augmente, la mesure d'exhaustivité augmente. Mais, cette fois-ci, l'augmentation n'est pas linéaire et on peut constater les sauts et des paliers dans la courbe. Cela peut être expliqué par le fait l'homogénéité de l'écriture dans la base « Washington » est moins élevée que celle de l'écriture dans les bases « Saint Gall » et « Parzival ».

Selon les résultats obtenus, nous pouvons constater que la mesure d'exhaustivité a une tendance d'augmenter lorsque le nombre d'invariants augmente. À l'extrême, dans le cas où les invariants sont touts les *strokes* extraits, la mesure d'exhaustivité des invariants : $E(X) \approx 1$. Mais, lorsque le nombre d'invariants est élevé, nous avons plus de redondance entre les invariants. Cette redondance rend fastidieuse pour l'utilisateur la composition de requête, et rend le système de recherche peu stable. Donc, la mesure d'exhaustivité n'est pas suffisante, à elle seule, pour évaluer la qualité des invariants.

Discussion :

Nous avons présenté dans cette section la mesure d'exhaustivité, une caractéristique importante dans l'évaluation de la qualité des invariants, puis qu'elle mesure la capacité d'un ensemble d'invariants à représenter de manière fidèle un ensemble de mots. Nous présentons le calcul de mesure d'exhaustivité dans la section 4.3.1.1 Nous présentons également 2 expérimentations dans la section 4.3.1.2 et la section 4.3.1.3. Le résultat de la première expérimentation montre que la mesure d'exhaustivité que nous proposons est cohérente avec son objectif et avec nos expérimentations précédentes sur la recherche de mots. Le résultat de la deuxième expérimentation montre que nous ne pouvons pas utiliser seulement la mesure d'exhaustivité pour évaluer la qualité des invariants, car cette mesure a tendance à favorise des invariants en grand nombre (donc redondants).

Pour cette raison, nous devons utiliser une autre mesure : « l'unicité », qui évalue la redondance des invariants. Nous présentons la mesure d'unicité dans la section suivante.

4.3.2. Mesure d'unicité

L'unicité évalue la redondance des invariants. Pour la composition de requêtes, la redondance des invariants rendent fastidieuse pour l'utilisateur la composition de requête. Pour la recherche de mots, la redondance rend le système de recherche peu robuste à la variation de l'écriture. Donc, la mesure de l'unicité est essentielle pour évaluer la qualité des invariants. Dans cette section, nous présentons la méthode d'évaluation que nous proposons pour mesurer l'unicité des invariants.

L'idée principale de notre méthode est de mesurer la dissimilarité entre les différentes représentations d'un même mot obtenues, en utilisant les invariants extraits. Si la redondance entre les invariants est élevée, alors la dissimilarité entre les représentations d'un même mot est élevée. Pour plus de détails sur le calcul de la mesure d'unicité, nous renvoyons le lecteur à la section 4.3.2.1.

Cette section consiste en 2 parties. Dans la première partie, nous proposons une méthode pour mesurer l'unicité des invariants. La deuxième partie montre notre expérimentation pour valider la cohérence de la mesure d'unicité que nous proposons.

4.3.2.1. Calcul de la mesure d'unicité

Pour calculer la mesure d'unicité des invariants extraits, nous utilisons une base de vérité terrain qui contient les images de mots de la collection de documents avec leurs annotations. Les images de mots dans cette base sont regroupées en classes. Chaque classe contient les images d'un même mot. Puis, pour chaque image, nous prenons les invariants préalablement extraits de la base pour caractériser cette image en utilisant la méthode présentée dans la section 4.2.1 (représentation basée sur graphe). Ensuite, pour chaque classe, nous calculons l'hétérogénéité dans cette classe. Nous utilisons la méthode présentée dans la section 4.2.2 pour calculer la distance entre chaque paire de représentations d'images d'une même classe. Puis, nous calculons ensuite la distance moyenne à l'intérieur de chaque classe C (mesure d'hétérogénéité) :

$$dist_C = \frac{2}{N_c(N_c-1)} \sum_{k=1}^{N_c-1} \sum_{m=k+1}^{N_c} dist(I_k^C, I_m^C)$$

Où N_c est le nombre des images de mot dans la classe $C = \{I_k^C | k = 1..N_c\}$

Ensuite, nous calculons la compacité de la classe C :

$$comp_C = e^{-\beta * dist_C}$$

Où β est une constante. Dans notre application, nous choisissons heuristiquement $\beta = 20$.

La mesure d'unicité des invariants $U(X)$ pour une base de vérité-terrain donnée est la médiane de la compacité des classes d'images dans la base de vérité terrain.

Nous avons présenté dans cette section le calcul de la mesure d'unicité. Nous pouvons utiliser cette mesure pour évaluer la redondance des invariants. Dans la section suivante, nous présentons une expérimentation pour vérifier expérimentalement la cohérence de la mesure d'unicité proposée.

4.3.2.2. Expérimentation : Comparaison des mesures d'unicité des différents invariants extraits par la méthode de clustering K-moyennes avec des différents paramètres k

L'objectif de cette expérimentation est de vérifier la cohérence de la mesure d'unicité que nous proposons. Plus précisément, nous réalisons l'expérimentation sur des bases de données suivantes : Base « Saint Gall », base « Washington », base « Parzival ». Pour chaque base de données, nous extrayons des *strokes* en utilisant

la configuration de regroupement indiquée dans la section 3.3.2. Puis, nous appliquons la méthode de pré-*clustering* présentée dans la section 3.4.1 pour pré-segmenter les *strokes* en 4 *clusters*. Puis, pour chaque *cluster*, nous appliquons une méthode de *clustering* K-moyenne sur les *strokes* de ce *cluster* en modifiant le paramètre K de 2 à 50 pour générer différents ensembles d'invariants de tailles croissantes. Ensuite, nous faisons la comparaison des mesures d'unicité de ces ensembles d'invariants.

Les figures : Figure 4.16, Figure 4.18, Figure 4.17 montrent les résultats de notre expérimentation sur les bases : « Saint Gall », « Washington » et « Parzival » (voir section 4.2.3.1). Chaque figure montre aussi la mesure d'unicité des invariants qui sont extraits en utilisant la méthode de « consensus *clustering* » (présentée dans la section 3.4.3).

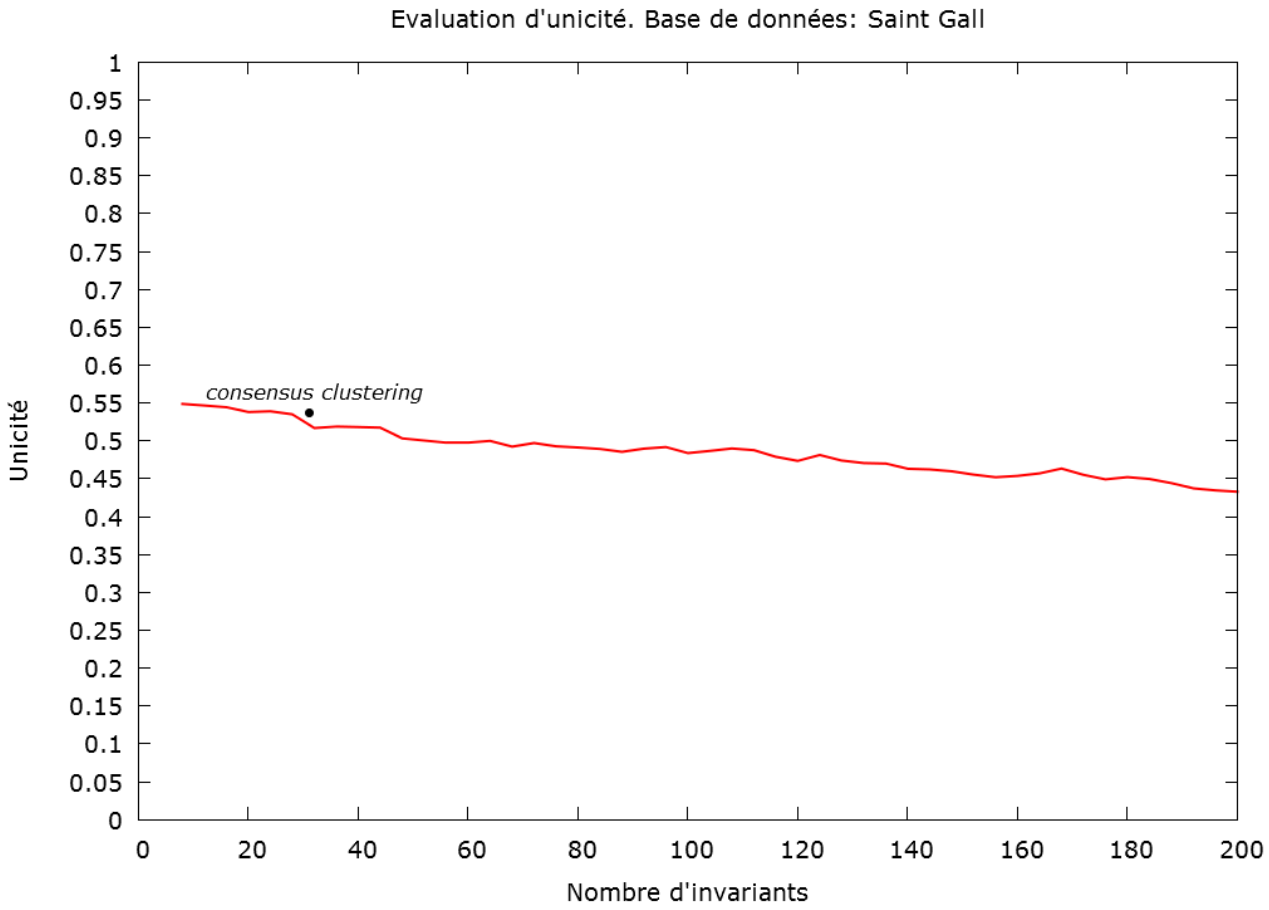


Figure 4.16 : Mesures d'unicité des ensembles d'invariants extraits de la base « Saint Gall »

Nous pouvons constater que :

- Lorsque le nombre d'invariants augmente, la mesure d'unicité diminue de manière linéaire.
- Grâce à la bonne qualité des documents dans la base « Saint Gall » et à l'homogénéité élevée de l'écriture (écriture bien soignée, mono-scripteur), généralement, la mesure d'unicité des invariants extraits de la base « Saint Gall » est élevée.

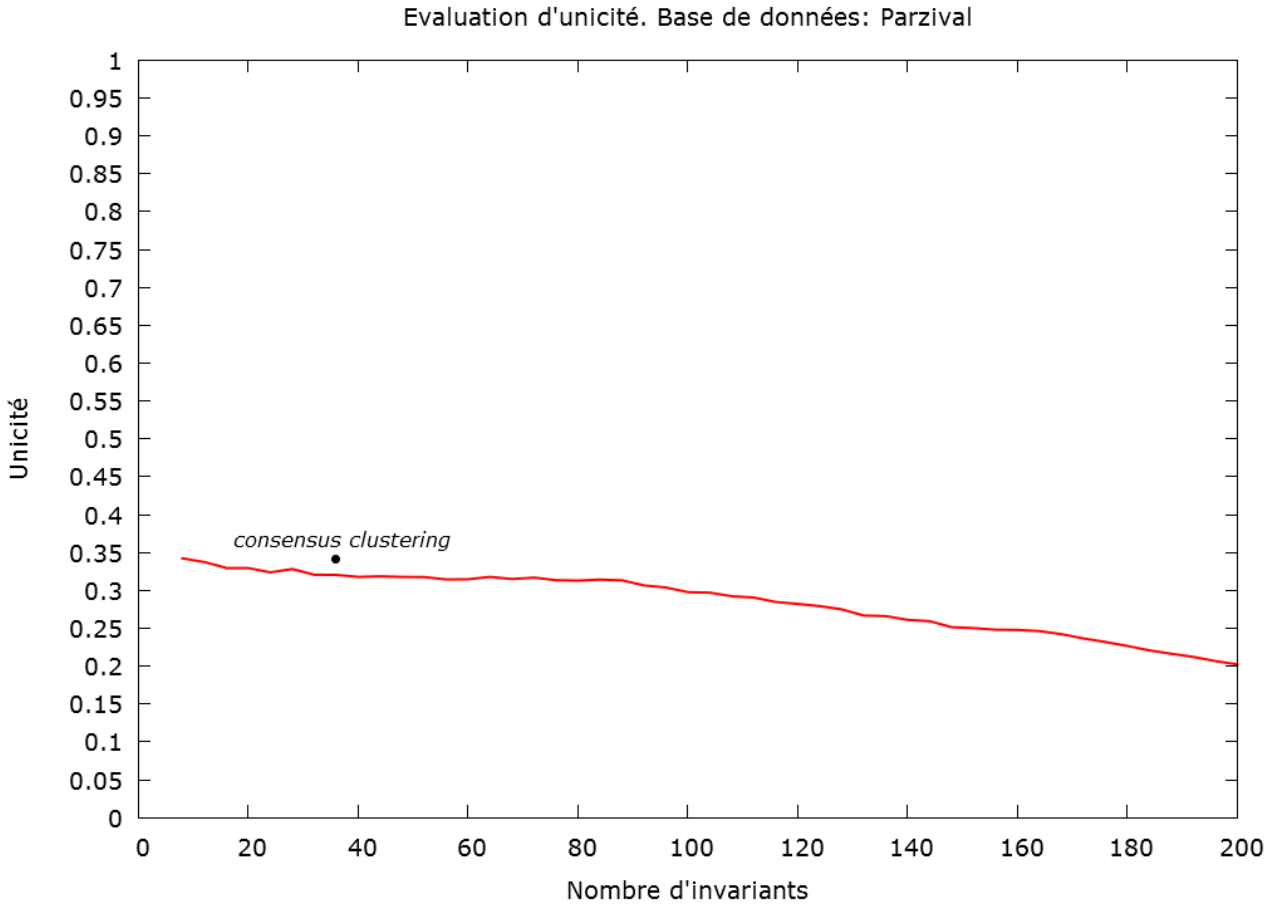


Figure 4.17 : Mesures d'unicité des ensembles d'invariants extraits de la base « Parzival »

Nous pouvons constater que :

- Lorsque le nombre d'invariants augmente, la mesure d'unicité diminue, mais reste plus faible que celle de la base « Saint Gall » du fait de la moins bonne qualité de la base « Parzival ».

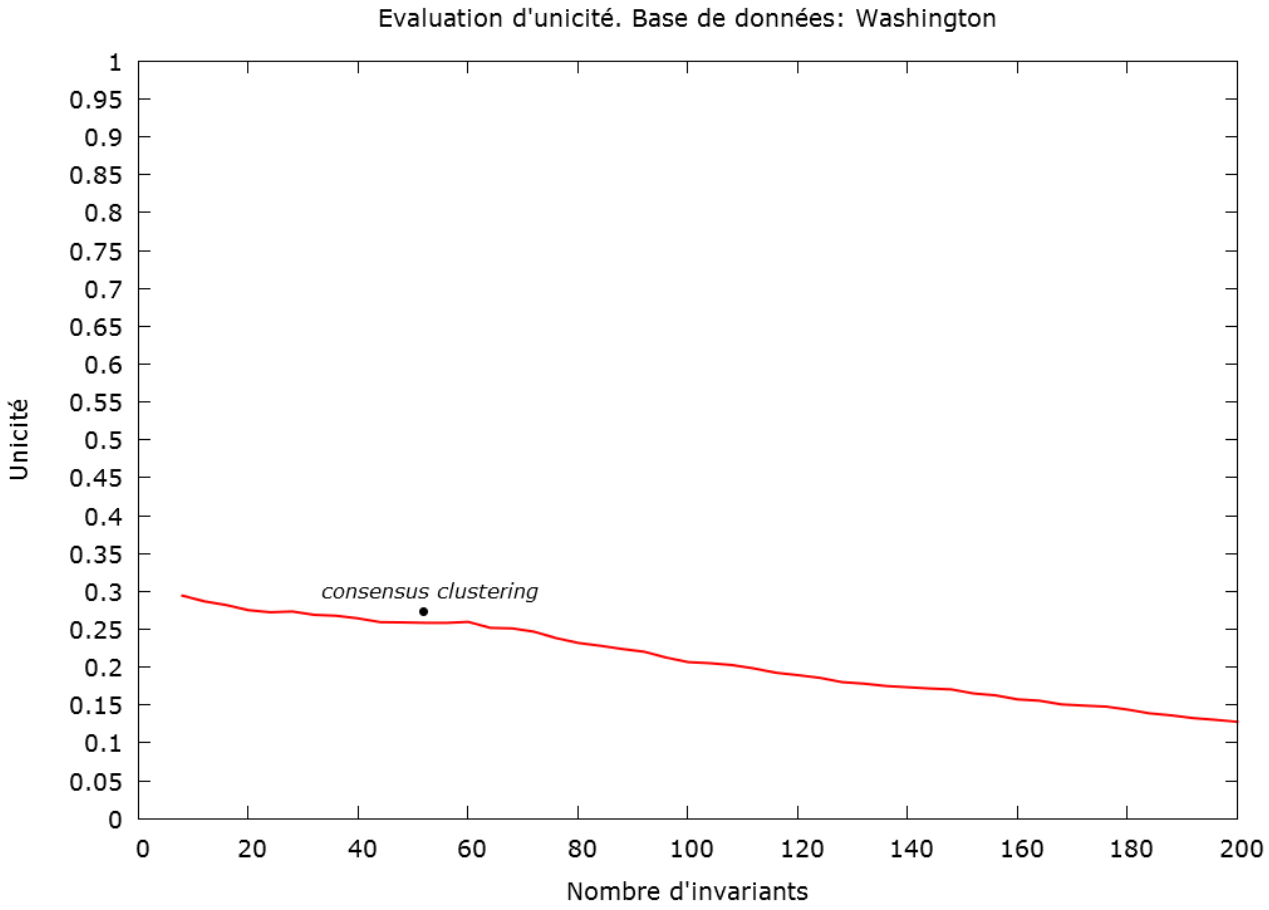


Figure 4.18 : Mesures d'unicité des ensembles d'invariants extraits de la base « Washington »

Nous pouvons constater que :

- Lorsque le nombre d'invariants augmente, la mesure d'unicité diminue de manière linéaire.
- L'homogénéité de l'écriture dans la base « Washington » est moins élevée que celle de l'écriture dans les bases « Saint Gall » et « Parzival ». Généralement, la mesure d'unicité des invariants extraits de la base « Washington » est moins élevée que celles des invariants extraits des bases « Saint Gall » et « Parzival ».

Selon le résultat que nous obtenons, nous pouvons constater que :

- La mesure d'unicité a tendance à diminuer lorsque le nombre d'invariants augmente.
- Les invariants extraits des bases de données dont l'homogénéité est élevée (mono-scripteur, écriture soignée) ont une mesure d'unicité élevée.

Ces faits peuvent s'expliquer de la manière suivante :

- Lorsque le nombre d'invariants augmente, le nombre d'invariants similaire augmente (c'est-à-dire, nous avons plus de redondance).
- L'écriture dans des bases de données d'homogénéité élevée a une plus faible variation que celle des autres bases de données. Donc, dans le cas des bases de données de l'homogénéité élevée, plus de *strokes* similaires sont regroupés dans un même *cluster* lors du *clustering* de *strokes*. Les invariants extraits ont alors moins de redondance que ceux des autres bases de données.

La mesure d'unicité est donc cohérente pour mesurer la redondance des invariants.

4.3.3. Mesure de qualité

Nous avons présenté dans les sections 4.3.1 et 4.3.2 les mesures d'exhaustivité et d'unicité des invariants. Ces deux mesures sont essentiellement importantes pour l'évaluation de la qualité des invariants. Nous souhaitons à présent combiner ces 2 mesures en une mesure unique pour évaluer des invariants. Dans cette section, nous présentons la mesure de qualité, qui est une combinaison des mesures d'exhaustivité et d'unicité, pour évaluer les invariants. Puis, nous présentons une expérimentation qui vérifie la cohérence de la mesure de qualité que nous proposons.

4.3.3.1. Calcul de la mesure de qualité

Supposons que nous avons un ensemble d'invariants X. $exhaust(X)$ est la mesure d'exhaustivité de X. $uni(X)$ est la mesure d'unicité de X. $quali(X)$ est la mesure de qualité de X, qui est la combinaison de $exhaust(X)$ et $uni(X)$, calculée comme suite :

$$quali(X) = squash\big(exhaust(X)\big) * uni(X)$$

Où $squash(x)$ est une fonction sigmoïde, définie comme suit :

$$squash(x) = \frac{2}{\pi} * \text{atan}\left(\frac{\pi}{2} * \beta * x\right)$$

Où β est une constante. Dans notre application, nous choisissons heuristiquement $\beta = 10$.

Dans cette section, nous avons présenté le calcul de la mesure de qualité. Dans la section suivante, nous présentons une expérimentation pour vérifier la cohérence de cette mesure.

4.3.3.2. Expérimentation : Comparaison des mesures de qualité des différents ensemble d'invariants extraits par la méthode de clustering K-moyenne

Nous présentons dans cette section une expérimentation pour vérifier la cohérence de la mesure de qualité que nous proposons. Plus précisément, nous réalisons l'expérimentation sur des bases de données suivantes : Base « Saint Gall », base « Washington » et base « Parzival » (voir section 4.2.3.1). Pour chaque base de données, nous extrayons des *strokes* en utilisant la configuration de regroupement indiquée dans la section 3.3.2. Puis, nous appliquons la méthode de pré-*clustering* présentée dans la section 3.4.1 pour pré-segmenter les *strokes* en 4 *clusters*. Puis, pour chaque *cluster*, nous appliquons une méthode de *clustering* K-moyennes sur les *strokes* de ce *cluster* en modifiant le paramètre K de 2 à 50 pour générer différents ensembles d'invariants de tailles croissantes. Ensuite, nous faisons la comparaison des mesures de qualité de ces ensembles d'invariants.

Les figures : Figure 4.19, Figure 4.21, Figure 4.20 montre le résultat de notre expérimentation sur les bases de données « Saint Gall », « Washington » et « Parzival » respectivement. Chaque figure montre aussi la mesure de qualité des invariants qui sont extraits en utilisant la méthode de « consensus *clustering* » (présentée dans la section 3.4.3).

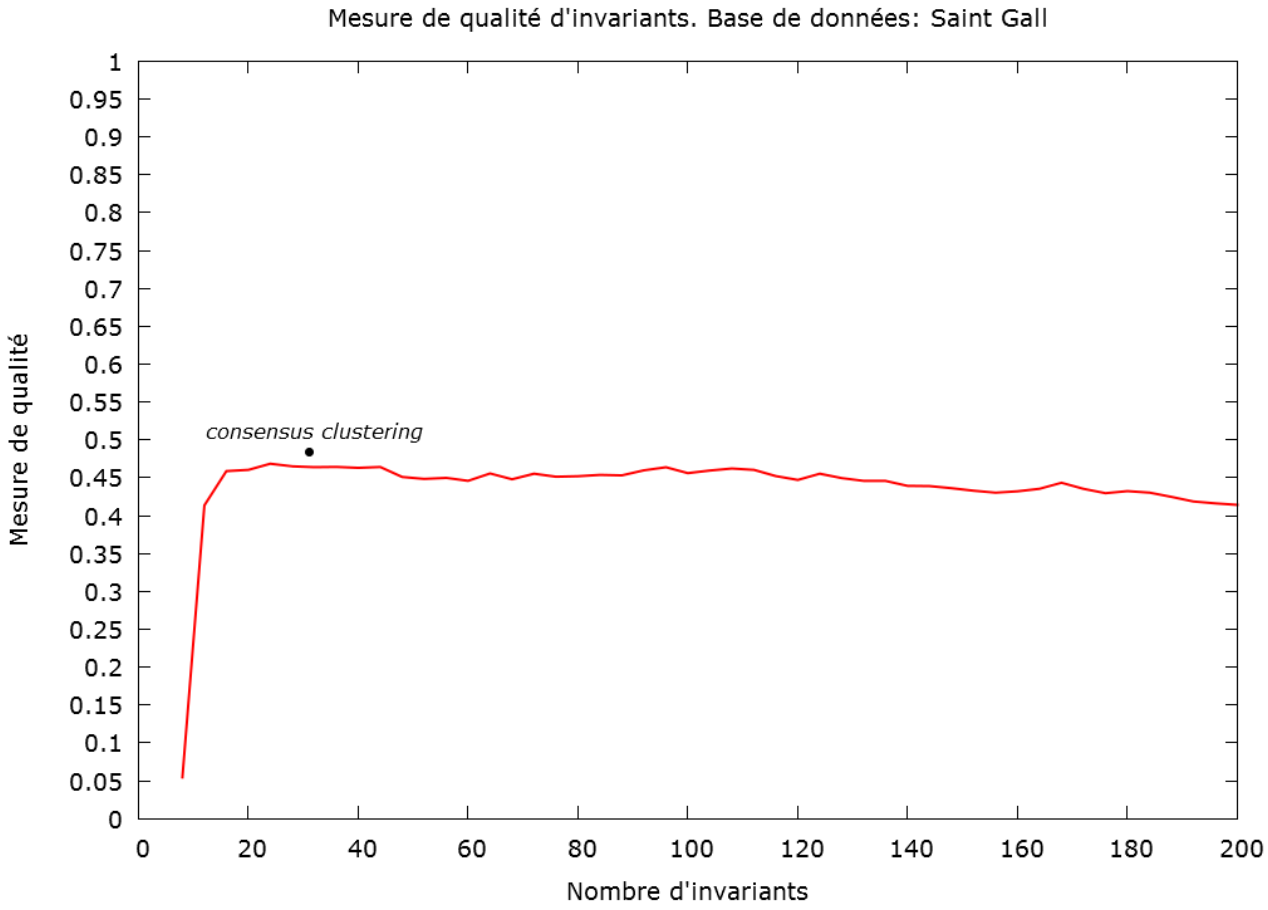


Figure 4.19 : Mesures de qualité des ensembles d'invariants extraits de la base Saint Gall

Nous pouvons constater que la mesure de qualité augmente jusqu'à ce qu'elle atteigne une valeur maximale $quali(I) = 0{,}455$ avec le nombre d'invariants $n = 24$, et puis, la mesure de qualité diminue lorsque le nombre d'invariants augmente.

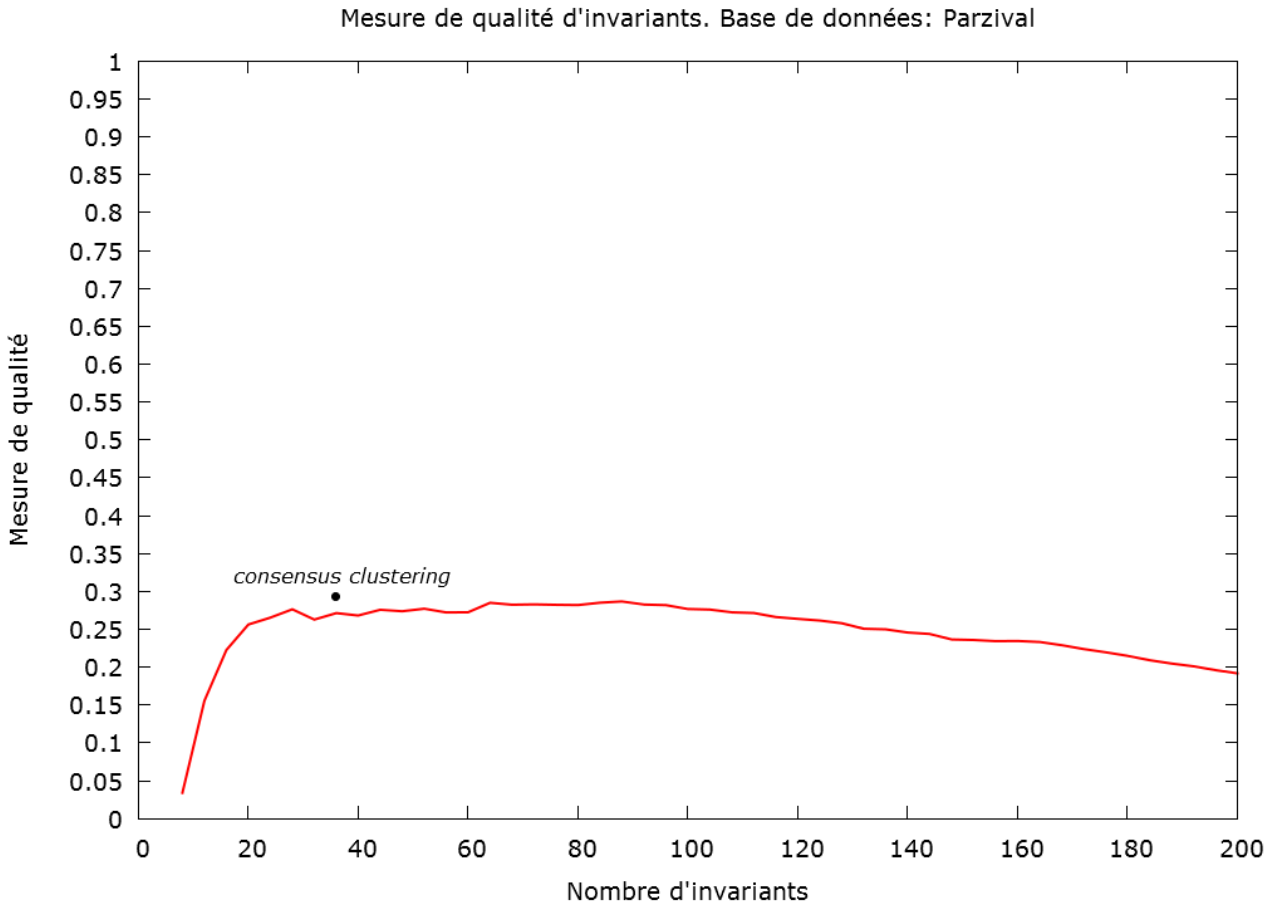


Figure 4.20 : Mesures de qualité des ensembles d'invariants extraits de la base Parzival

Nous pouvons constater que la mesure de qualité augment jusqu'elle atteint une valeur maximale $quali(I)$ =0.29 avec le nombre d'invariants $n = 84$, et puis, la mesure de qualité diminue lorsque le nombre d'invariants augmente.

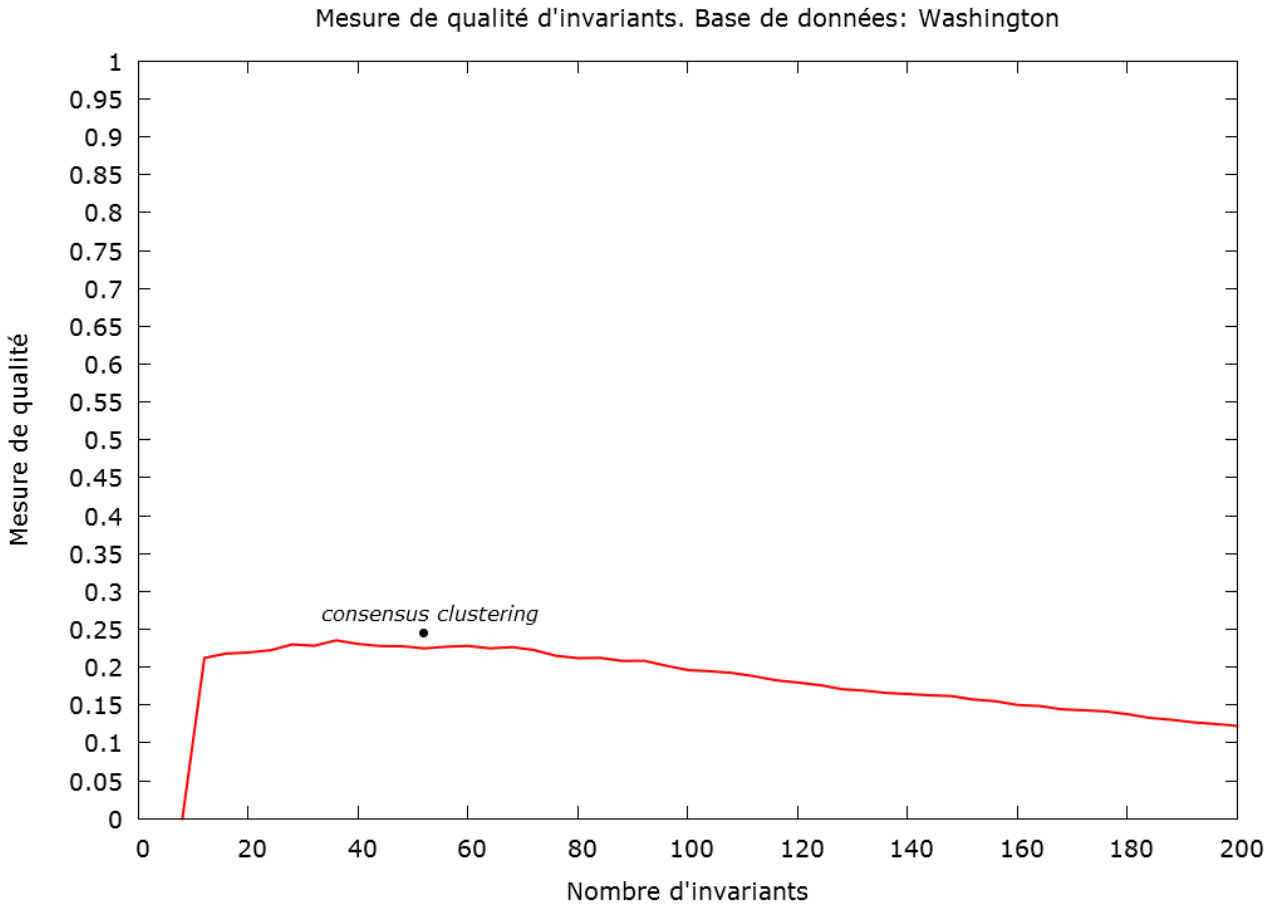


Figure 4.21 : Mesures de qualité des ensembles d'invariants extraits de la base Washington

Nous pouvons constater que la mesure de qualité augment jusqu'à ce qu'elle atteigne une valeur maximale $quali(I) = 0.24$ avec le nombre d'invariants $n = 36$, et puis, la mesure de qualité diminue lorsque le nombre d'invariants augmente.

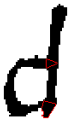
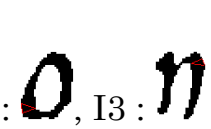
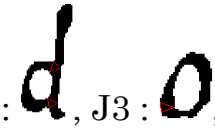
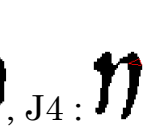
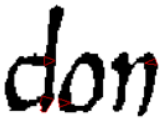
Dans les trois cas, la mesure de qualité proposée atteint son maximum lorsqu'un bon compromis entre exhaustivité (et donc la capacité des invariants à représenter les mots) et unicité (et donc la redondance des invariants) est trouvée. On peut remarquer sur les trois figures précédentes que la mesure de qualité atteint son maximum aux alentours du nombre de clusters retenus par le consensus *clustering*.

4.3.4. Conclusion

Nous avons présenté dans la section 4.3 une méthode pour évaluer la qualité des invariants extraits à partir d'une collection de documents. Nous ne pouvons pas évaluer la sémantique des invariants, à cause de la difficulté dans l'obtention

d'une vérité terrain qui juge la pertinence des invariants extraits (le coût important de la ressource humaine, le jugement d'une personne est objectif, *etc.*). Par contre, nous présentons 2 mesures : la mesure d'exhaustivité et la mesure d'unicité. La mesure d'exhaustivité indique la capacité des invariants de représenter toutes les images de mots dans la base de données, tandis que la mesure d'unicité indique la redondance des invariants. Ces deux mesures sont essentielles pour l'évaluation de la qualité des invariants.

Nous avons présenté dans les sections 4.3.1.1 et 4.3.2.1 le calcul de la mesure d'exhaustivité et de la mesure d'unicité. Nous donnons ici un exemple pour montrer la cohérence de ces 2 mesures :

- Supposons que nous avons 2 ensembles d'invariants :
  - Ensemble 1 : $X1$ contient 3 invariants : I1 :  , I2 :  , I3 : 
  - Ensemble 2 : $X2$ contient 4 invariants : J1 :  , J2 :  , J3 :  , J4 : 
- Supposons que la base de données contient 2 images : Image 1 : 
  et Image 2 : 
- La mesure d'exhaustivité des invariants de l'ensemble 1 : $E(X1) = 0{,}9$. La mesure d'exhaustivité des invariants de l'ensemble 2 : $E(X2) = 1$.
- Nous pouvons constater que l'image 2 dans la base de données n'est pas parfaitement décrite par les invariants de l'ensemble 1. La meilleure description possible par les invariants de l'ensemble 1 est : $\{I1, I2, I3\}$. Par contre, l'image 2 est parfaitement décrite par les invariants de l'ensemble 2 en utilisant la composition : $\{J2, J3, J4\}$. Donc, les invariants de l'ensemble 2 décrit les images dans la base de données mieux que les invariants de l'ensemble 1. Dans le calcul de mesure d'exhaustivité des 2 ensembles d'invariants présenté ci-dessus, nous avons : $E(X2) > E(X1)$. Donc, la mesure d'exhaustivité que nous proposons est cohérente pour mesurer l'information décrite par les invariants.
- Par contre, la mesure d'unicité des invariants de l'ensemble 1 : $U(X1) = e^0 = 1$. La mesure d'unicité des invariants de l'ensemble 2 : $U(X2) =$

$e^{-20*\left(\frac{1}{18}\right)} = 0{,}329 < U(X1)$, car les invariants de l'ensemble 2 sont plus redondants que ceux de l'ensemble 1.

- La mesure de qualité des invariants de l'ensemble 1 : $Q(X1) = \frac{2}{\pi} * \text{atan}\left(\frac{\pi}{2} * 10 * E(X1)\right) * U(X1) = 0{,}96$. La mesure de qualité des invariants de l'ensemble 2 : $Q(X2) = \frac{2}{\pi} * \text{atan}\left(\frac{\pi}{2} * 10 * E(X2)\right) * U(X2) = 0{,}32 < Q(X1)$
- Nous pouvons constater que les invariants de l'ensemble 1 décrivent presque parfait les images dans la base de données. Les invariants de l'ensemble 2 décrivent parfait les images dans la base de données, mais ils ont plus de redondance que les invariants de l'ensemble 1 (les invariants J1 et J2 sont similaires). Les invariants de l'ensemble 2 ont donc la qualité inférieure que les invariants de l'ensemble 1.

Selon cet exemple et les expérimentations présentées en sections 4.3.1 et 4.3.2, nous pouvons donc conclure que la mesure d'exhaustivité et la mesure d'unicité que nous proposons sont cohérentes pour l'évaluation de la qualité des invariants.

Lorsque la mesure d'exhaustivité et la mesure d'unicité sont essentiellement importantes pour l'évaluation de la qualité des invariants, nous souhaitons à présent combiner ces 2 mesures en une mesure unique pour évaluer des invariants. Dans la section 4.3.3, nous présentons la mesure de qualité, qui est une combinaison des mesures d'exhaustivité et d'unicité, pour évaluer les invariants. Le résultat d'expérimentation dans la section 4.3.3 confirme la cohérence de la mesure de qualité que nous proposons.

Chapitre 5 : Vers la composition interactive de requêtes pour la recherche de mots

5.1. Introduction

Dans le chapitre 2, nous avons présenté les systèmes de recherche de mots existants, leurs avantages et leurs inconvénients. Afin de contourner les inconvénients des systèmes existants, nous proposons un système générique, omni langage et interactif de recherche de mots dans les collections de documents (voir section 2.4). Nous avons donc conçu un système original basé sur l'extraction de formes récurrentes dans le texte (appelés « invariants »). La recherche des mots correspondant à la requête de l'utilisateur se fera à partir d'une signature structurelle calculée à partir de l'apparence et de l'agencement spatial des invariants. Pour formuler sa requête, l'utilisateur crée le mot à rechercher par la composition interactive de requête en utilisant les invariants extraits. L'idée est de proposer à l'utilisateur une interface intuitive qui lui permette de composer sa requête quand il n'a pas pu/voulu trouver une occurrence du mot à rechercher dans le document (et donc que le word spotting traditionnel ne peut être mis en œuvre). La Figure 5.1 montre un exemple de la composition de requête. Parmi les invariants extraits, l'utilisateur prend les invariants I1, I2, I3 et I4 (A) pour composer l'image du mot présentée dans (B).

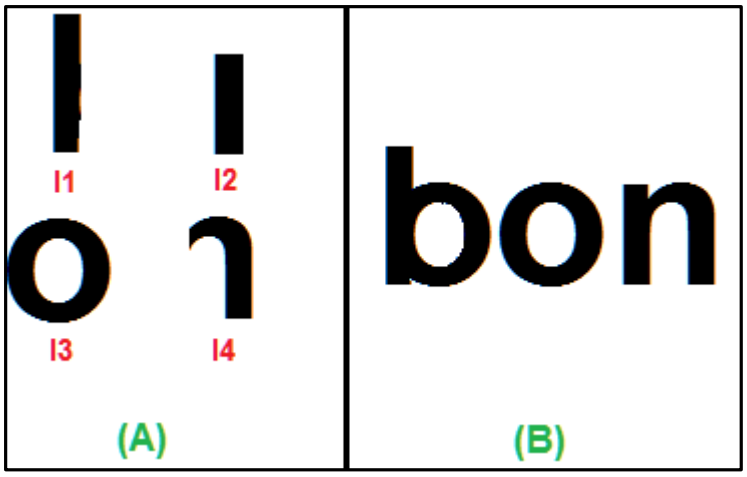


Figure 5.1 : Composition de requête : Utilisateur prend les invariants I1, I2, I3 et I4 (A) pour composer l'image du mot « bon » (B)

Pour que l'interface permettant à l'utilisateur de composer sa requête soit conviviale, il faut extraire les invariants qui font sens pour un humain (qui connait suffisamment le script et le langage utilisé pour juger de leur pertinence). Nous

avons présenté dans le chapitre 3 la méthode automatique d'extraction de ces invariants. Nous essayons d'extraire les invariants de manière à ce qu'ils aient un intérêt pour la composition de requêtes. Mais, lorsque le résultat de la méthode d'extraction d'invariants ne satisfait pas l'utilisateur, nous proposons une méthode de raffinements interactifs des invariants que nous présenterons dans la section 5.2. Grâce à cette méthode, la composition de requête est plus intuitive et conviviale pour l'utilisateur : elle a plus de sens.

5.2. Contribution 5 : Raffinements interactifs des invariants

Lorsque le résultat de la méthode d'extraction d'invariants ne satisfait pas l'utilisateur, nous proposons un module qui permet à l'utilisateur de raffiner le résultat. Nous définissons une taxonomie des raffinements selon 2 catégories :

- ***Raffinements dans l'espace de caractéristiques :*** Ces raffinements permettent à l'utilisateur de modifier les *clusters* de *strokes*. L'utilisateur peut fusionner les *clusters* des *strokes* similaires ou bien découper un *cluster* en sous-*clusters* qui soient plus homogènes.
- ***Raffinement dans l'espace spatial :*** Ces raffinements permettent à l'utilisateur de découper un *stroke* de grande taille (qui est souvent produit par un faut négatif lors de la détection de zones ambiguës) en des *strokes* de taille plus petite. Ils permettent également à l'utilisateur de regrouper des *strokes* de petite taille un plus grand *stroke* (ce qui peut avoir un intérêt pour la composition de requêtes par un humain en diminuant le nombre d'invariants nécessaires pour composer une requête).

Dans les sections suivantes, nous présentons nos méthodes de raffinement selon ces 2 catégories.

5.2.1. Méthode de raffinement interactif dans l'espace de caractéristiques

Nous avons présenté dans la section 3.4, la méthode pour l'extraction d'invariants par *clustering* de *strokes*. Bien sûr, les résultats du *clustering* ne sont pas satisfaisants notamment lors que :

- La variation d'un *cluster* est élevée, c'est-à-dire que plusieurs *strokes* d'apparences différentes sont dans un même *cluster*.
- Des *strokes* d'apparences similaires sont séparés dans plusieurs *clusters* différents.

Il est donc nécessaire de fournir une méthode qui permet de modifier le résultat de *clustering* afin d'améliorer la qualité des invariants extraits (les invariants étant définis, pour rappel, comme les prototypes des clusters). Nous proposons donc une méthode interactive qui permet à l'utilisateur de modifier le résultat de *clustering*. La Figure 5.2 montre le processus de raffinement dans l'espace de caractéristiques :

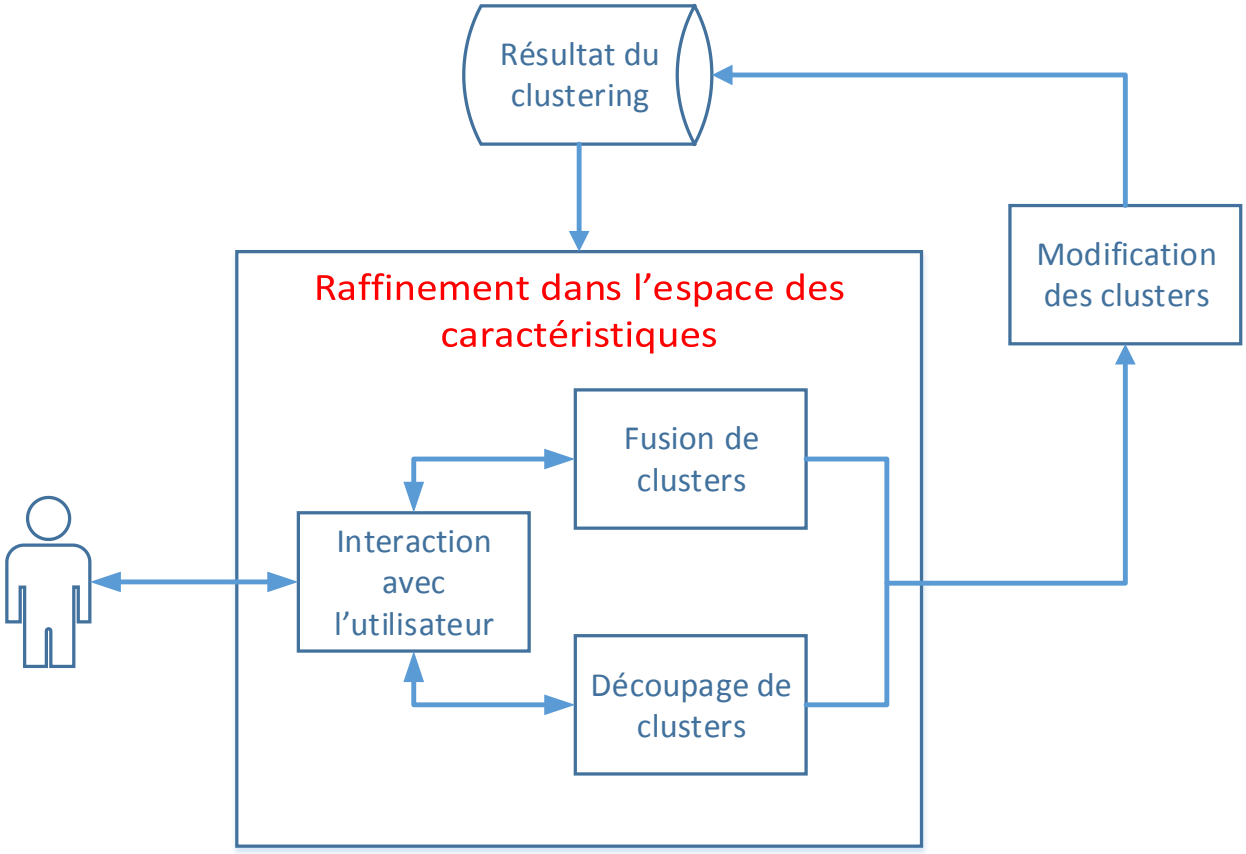


Figure 5.2 : Aperçu du processus de raffinements dans l'espace des caractéristiques

Plus précisément, le processus de raffinement présente le résultat du *clustering* à l'utilisateur. L'utilisateur peut réaliser les 2 opérations suivantes :

- *Fusion de clusters :* permet à l'utilisateur de fusionner plusieurs *clusters* similaires en un seul *cluster*.
- *Découpage de clusters :* permet à l'utilisateur de découper un *cluster* en plusieurs nouveaux *clusters* qui sont plus homogènes.

La Figure 5.3 montre l'interface qui permet à l'utilisateur de faire des raffinements dans l'espace de caractéristiques :

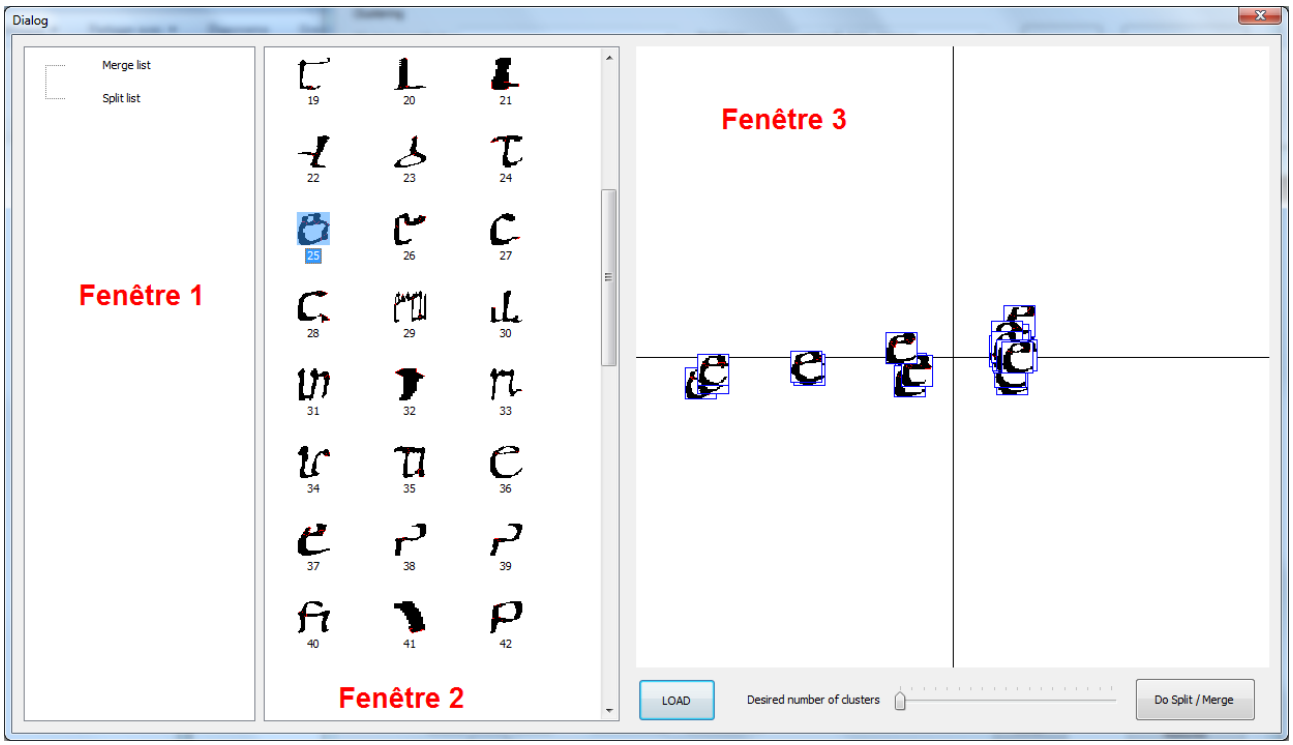


Figure 5.3 : L'interface de raffinements dans l'espace de caractéristiques

La fenêtre 2 montre les invariants extraits. Lorsque l'utilisateur clique sur un invariant dans la liste, les *strokes* dans le *cluster* de cet invariant sont affichés dans la fenêtre 3. Cette fonction permet à l'utilisateur d'analyser les *clusters* des invariants. Suite à son analyse, l'utilisateur choisit les *clusters* qu'il veut raffiner (fusion de *clusters*, découpage de *clusters*). Les commandes de l'utilisateur sont visualisées dans la fenêtre 1. Puis, le système fait des modifications de *clusters* et présente le résultat à l'utilisateur. L'utilisateur peut répéter ce processus tant qu'il n'est pas satisfait.

Dans les sections suivantes, nous présentons en détail les opérations de fusion de *clusters* et de découpage de *clusters*.

5.2.1.1. Fusion de clusters

La fusion de *clusters* permet à l'utilisateur de fusionner plusieurs *clusters* similaires en un seul *cluster*. La Figure 5.4 montre l'interface qui permet à l'utilisateur de fusionner les clusters.

Figure 5.4 : L'interface de fusion de *clusters*

Pour fusionner les *clusters*, l'utilisateur choisit les invariants similaires à l'aide de la visualisation d'invariants dans la fenêtre 2. Puis, l'utilisateur regroupe les clusters d'apparences similaires en groupes. La fenêtre 1 montre les groupes d'invariants établis par l'utilisateur. Ensuite, le système va regrouper tous les *strokes* des *clusters* des invariants qui sont dans le même groupe et créer un nouveau *cluster*. Les anciens *clusters* sont retirés du système. Lorsqu'un nouveau *cluster* est créé, nous calculons le prototype de ce nouveau *cluster* comme nouvel invariant.

Nous donnons un exemple de la fusion des *clusters* :

La Figure 5.5 montre les *strokes* du *cluster* numéro 25 :  :

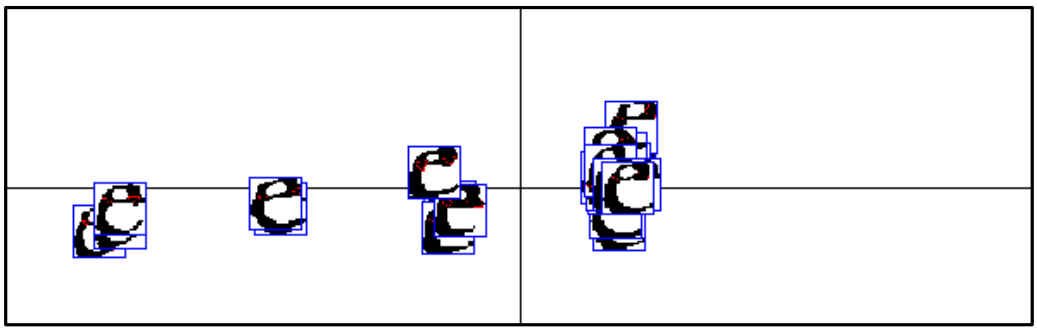


Figure 5.5 : Strokes dans le *cluster* 25

La Figure 5.6 montre les *strokes* du *cluster* numéro 37 :  :

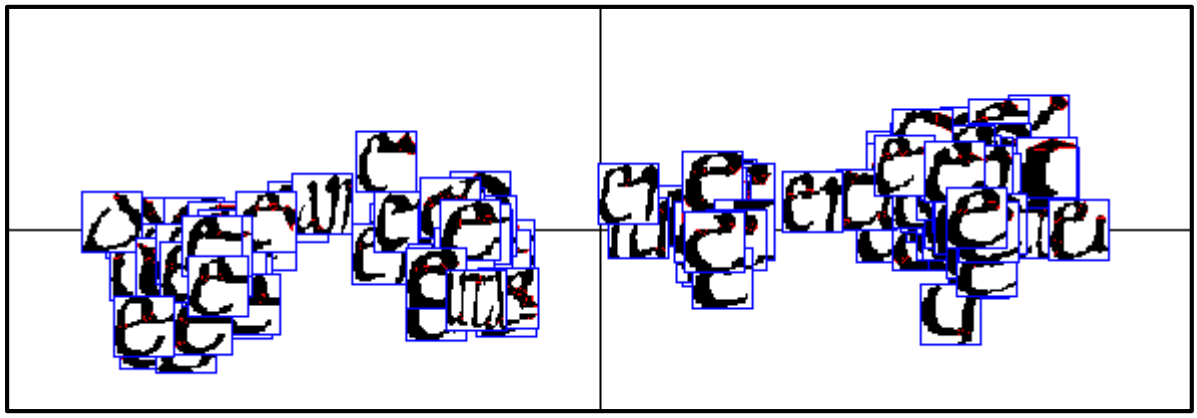


Figure 5.6 : Strokes dans le *cluster* 37

La Figure 5.7 montre le nouveau *cluster*, résultant de la fusion des *clusters* 25 et 37 :

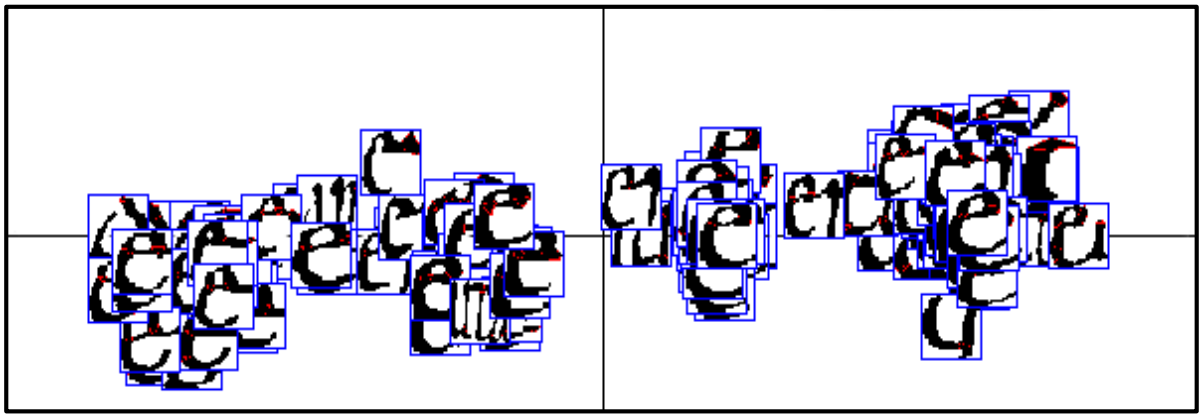


Figure 5.7 : Résultat de la fusion des *clusters* de *strokes* 25 et 37

Cette modification de la solution de *clustering* reste locale et n'a aucun impact sur les autres clusters ni sur les autres invariants.

5.2.1.2. Division de clusters

Cette opération permet à l'utilisateur de diviser un *cluster* en nouveaux *clusters* que l'on souhaite plus homogène. L'interface présentée dans la Figure 5.8 permet de découper des *clusters* :

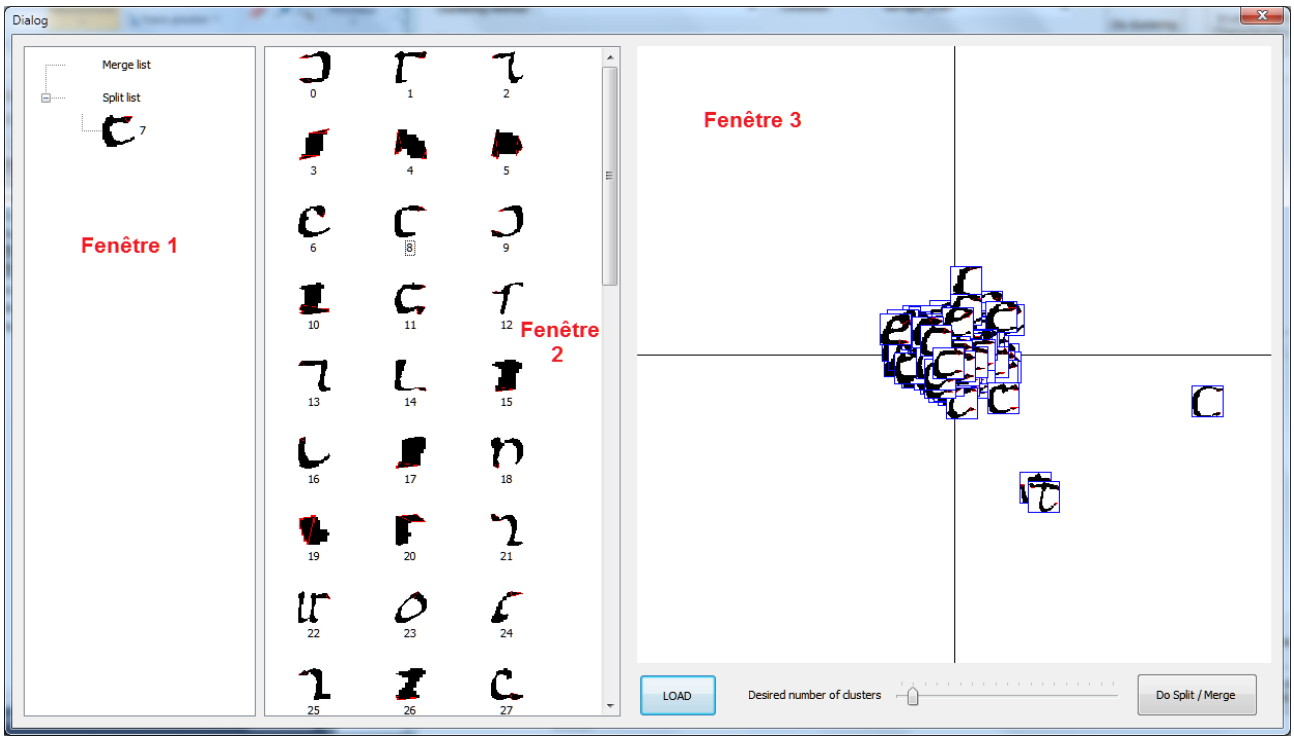


Figure 5.8 : L'interface de division de *clusters*

Dans l'exemple donné dans la Figure 5.8, le *cluster* numéro 7 (d'invariant ) n'est pas homogène (il est un mélange des *strokes* comme ). Il faut donc diviser ce *cluster* pour avoir des *clusters* plus homogènes. Plus précisément, l'utilisateur choisit le *cluster* C_i qu'il veut diviser, et le rajoute dans la « split list ». Puis, il indique le nombre de *clusters* qu'il veut obtenir en sortie : k. Ensuite, nous appliquons le *clustering* K-moyennes (voir l'annexe B.1) à partir des mêmes caractéristiques que précédemment avec le paramètre k sur les *strokes* du *cluster* C_i. Après le *clustering*, de nouveaux *clusters* sont créés. Lorsque des nouveaux *clusters* sont créés, nous calculons les prototypes de ces nouveaux *clusters* comme nouveaux invariants. Encore une fois, cette modification de la solution de *clustering* reste locale (limitée au *cluster* C_i) et n'a aucun impact sur les autres clusters (ni sur les autres invariants).

Nous donnons un exemple de la division des *clusters*. La Figure 5.9 montre ce *cluster* avant la division.

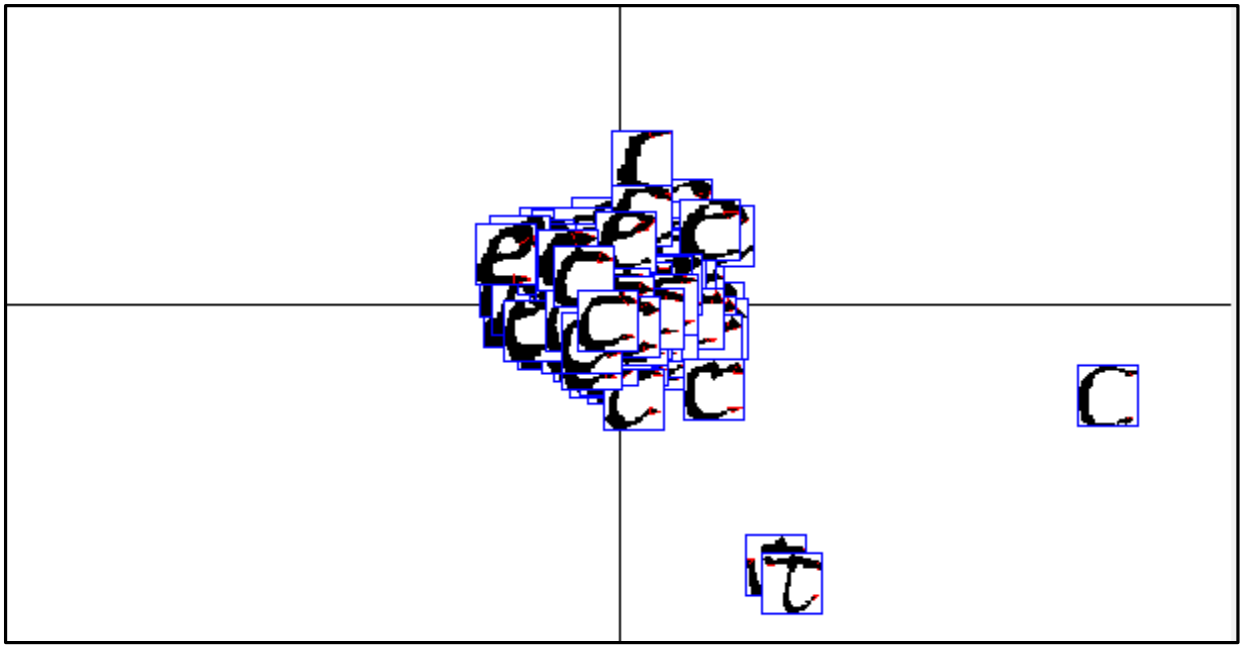


Figure 5.9 : Un *cluster* de *strokes* avant la division

La Figure 5.10, la Figure 5.11 et la Figure 5.12 montre 3 nouveaux *clusters* résultant du découpage.

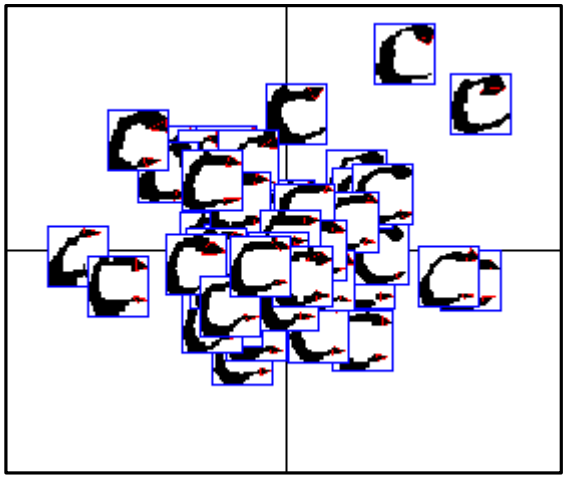


Figure 5.10 : Un *cluster* de *strokes* résultant de la division

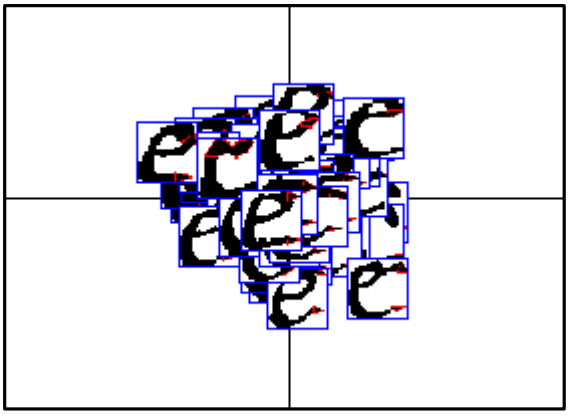


Figure 5.11 : Un *cluster* de *strokes* résultant de la division

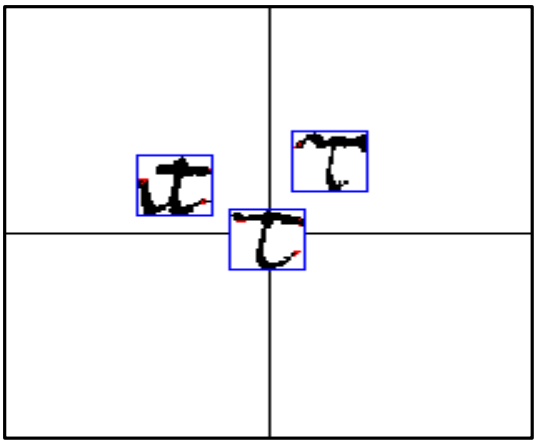


Figure 5.12 : Un *cluster* de *strokes* résultant

Nous pouvons constater qu'après le raffinement, nous obtenons des *clusters* plus homogènes.

Nous avons présenté la méthode de raffinement dans l'espace de caractéristiques dans les sections 5.2.1.1 et 5.2.1.2. Dans la section suivante (la section 5.2.1.3), nous présentons l'expérimentation pour comparer les invariants avant et après le raffinement dans l'espace de caractéristiques.

5.2.1.3. Expérimentation

Nous présentons dans cette section une expérimentation pour comparer les invariants avant et après le raffinement dans l'espace de caractéristiques. Plus précisément, nous réalisons l'expérimentation sur 3 bases de données : « Saint Gall », « Washington », « Parzival » (voir section 4.2.3.1). Pour chacune de ces bases de données, nous prenons un échantillon de la base pour extraire des invariants en appliquant la méthode d'extraction d'invariants présentée dans le chapitre 3.

Après l'extraction d'invariants, nous procédons manuellement un raffinement des invariants dans l'espace des caractéristiques. Le Tableau 5.1 montre le nombre de clics, le nombre d'opération réalisée, le temps de calcul total et le temps total au travers de l'interface homme-machine que nous avons développée pour faire ce raffinement :

Base de données	***Saint Gall***	***Washington***	***Parzival***
Nombre de clics	56	91	42
Nombre d'opérations réalisées dans l'espace des caractéristiques	14	23	11
Temps de calcul total	4 minutes	9 minutes	3 minutes
Temps de calcul moyen pour une opération	17 seconds	24 seconds	16 seconds
Temps total pour le raffinement	114 minutes	187 minutes	79 minutes

Tableau 5.1 : Convivialité et ergonomie de l'interface homme-machine pour un raffinement dans l'espace de caractéristiques

Nous obtenons les invariants après le raffinement. Puis, pour chaque base de données, nous utilisons la méthode d'évaluation d'invariants pour évaluer les invariants avant et après le raffinement.

Le Tableau 5.2 montre les mesures d'exhaustivité, d'unicité et de qualité des invariants avant et après le raffinement :

Base de données	***Saint Gall***		***Washington***		***Parzival***	
	Avant le raffinement	***Après le raffinement***	***Avant le raffinement***	***Après le raffinement***	***Avant le raffinement***	***Après le raffinement***
Nombre d'invariants	31	25	52	38	36	27
Mesure d'exhaustivité	**0,4019**	0,3942	**0,3750**	0,3657	**0,2915**	0,2849
Mesure d'unicité	0,5378	**0,5852**	0,2735	**0,2903**	0,3404	**0,3758**
Mesure de qualité	0,4840	**0,5255**	0,2442	**0,2584**	0,2938	**0,3232**

Tableau 5.2 : Mesures d'exhaustivité, d'unicité et de qualité des invariants avant et après le raffinement

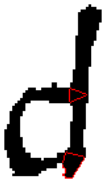
Nous pouvons constater que le raffinement entraine dans tous les cas une diminution du nombre d'invariants, la mesure d'exhaustivité des invariants diminue, tandis que leur mesure d'unicité augmente après le raffinement.

Sur les 3 bases de données, nous pouvons constater que la mesure de qualité des invariants est améliorée après le raffinement.

5.2.1.4. Discussion

Nous avons présenté dans cette section (la section 5.2.1) la méthode de raffinement interactif dans l'espace de caractéristiques. L'objectif de cette méthode est de modifier les *clusters* de *strokes* avec l'aide de l'utilisateur pour obtenir des invariants plus représentatifs. Nous avons calculé la mesure de qualité des invariants avant et après le raffinement et nous avons conclu que la qualité des invariants après le raffinement est améliorée. Néanmoins, le temps total pour faire un raffinement est assez long. Ce phénomène n'est pas causé par le calcul dans l'espace de caractéristiques (fusion de clusters, division de clusters) mais par les opérations sur le disque dur (récupérer les caractéristiques de *strokes*, récupérer la structure des clusters, enregistrer le résultat du raffinement, *etc.*). Le nombre de clics à réaliser pour une opération donnée est également important. Dans l'avenir, nous allons chercher à améliorer l'ergonomie de l'interface homme-machine que nous avons développée pour le raffinement dans l'espace de caractéristiques et réduire le temps des opérations sur le disque dur.

Par contre, la méthode de raffinement dans l'espace de caractéristiques ne peut pas remplir la fosse de sémantique des invariants qui est subjective et dépend de l'utilisateur. Nous donnons ici un exemple :

Le système extraits les invariants suivants (ensemble 1) : I1 :  , I2 :  et I3 :  .

Selon sa perception du langage et sa maîtrise de l'outil informatique, un utilisateur pourrait préférer manipuler ces invariants (ensemble 2) pour la composition de requêtes : J1 :  , J2 :  et J3 :  .

En effet, avec l'ensemble 2, l'invariant I1 peut être obtenu par composition des invariants J1 et J2, tandis que l'invariant J1 peut être réutilisé pour composer une grande variété de requêtes.

La méthode de raffinement dans l'espace de caractéristiques ne peut pas satisfaire ce souhait de l'utilisateur, parce qu'il fait seulement la modification de *clusters* des invariants sans modifier les *strokes*. Il faut donc une méthode qui modifie spatialement les *strokes* (avec l'intervention de l'utilisateur). Pour résoudre ce problème, nous proposons une méthode de raffinement interactif dans l'espace spatial que nous présentons dans la section suivante.

5.2.2. Méthode de raffinement interactif dans l'espace spatial

Pour obtenir les invariants, nous devons d'abord extraire des *strokes* à partir de la collection de documents. Nous avons présenté dans les sections 3.2 et 3.3 la méthode d'extraction de *strokes*. L'extraction de *strokes* repose sur une méthode d'extraction de zones ambiguës qui peut produire des faux positifs ou des faux négatifs et donc les *strokes* primaires produits sont parfois trop grands ou trop petits. Puisque nous utilisons une fonction générale pour décider de la continuité de 2 *strokes* primaires joignant une zone ambiguë, il est difficile de paramétrer celle-ci de manière à ce que le système donne de bons résultats avec tous les types de documents. En présence de certains types de langages ou détériorations, le système peut produire des *strokes* trop grands ou trop petits et ils ne sont pas toujours significatifs pour l'utilisateur.

De plus, puisque la sémantique des invariants est parfois subjective (nous renvoyons le lecteur à l'exemple dans la section 5.2.1.4), il est donc nécessaire de fournir une méthode qui permette de raffiner (avec l'intervention de l'utilisateur) le résultat d'extraction de *strokes* afin de satisfaire l'utilisateur. Nous présentons dans cette section une méthode interactive qui permet l'utilisateur de raffiner spatialement les *strokes* extraits. La Figure 5.13 montre le processus de raffinement de *strokes* dans l'espace spatial :

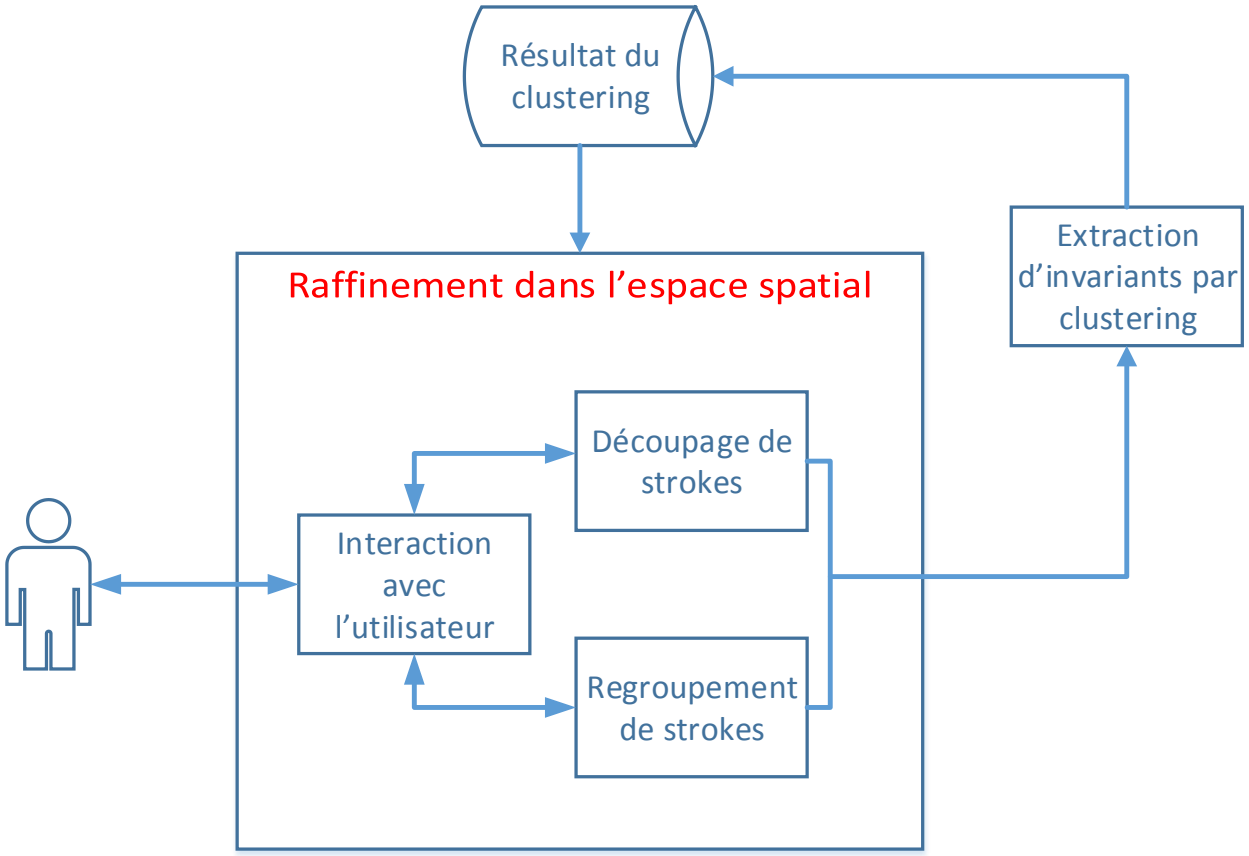


Figure 5.13: Processus de de raffinement dans l'espace spatial

Plus précisément, l'utilisateur peut réaliser les 2 opérations suivantes :

- Découpage de *strokes* : cette opération permet à l'utilisateur de découper un *stroke* en des *strokes* plus petits.
- Regroupement de *strokes* : cette opération permet à l'utilisateur de regrouper des *strokes* en un *stroke* plus grand.

Bien sûr, il n'est pas question ici de faire interagir l'utilisateur directement avec les *strokes* qui sont trop nombreux. Une originalité de notre proposition ici est donc de faire interagir l'utilisateur au niveau des clusters (invariants) et de répercuter ses interactions au plus bas niveau de l'extraction des *strokes*. Dans les sous-sections suivantes, nous présentons le fonctionnement de ces 2 opérations (découpage et regroupement).

5.2.2.1. *Découpage de strokes*

Le découpage de *strokes* est une opération qui permet à l'utilisateur de découper un *stroke* (dans l'espace spatial). La Figure 5.14 montre l'interface de l'opération « découpage de *strokes* » :

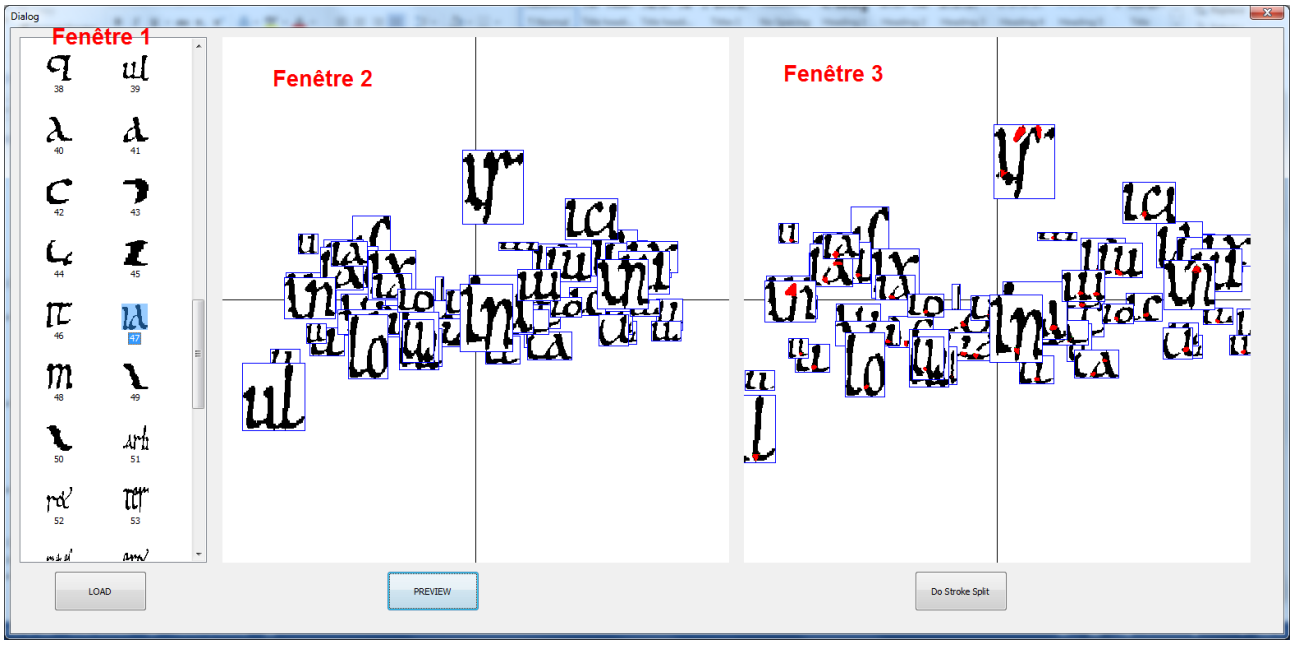


Figure 5.14 : L'interface de l'opération « Découpage de *strokes* »

La liste des invariants est présentée dans la fenêtre 1. L'utilisateur choisit l'invariant du *cluster* qu'il veut visualiser en vue de les découper. La fenêtre 2 montre les *strokes* dans le *cluster* de cet invariant. Ensuite et pour chaque *stroke* d'un *cluster* choisi, le système essaie de découper ce *stroke* en au moins 2 sous-*strokes*. Le résultat du découpage est pré-visualisé dans la fenêtre 3. Un point en rouge présenté dans la fenêtre 3 indique le point de découpage sélectionné.

La méthode pour découper un *stroke* est la suivante :

- Pour chaque *stroke* à découper, nous appliquons la méthode d'extraction de zones ambiguës présentée dans la section 3.2 en modifiant les paramètres de la méthode d'extraction de zones ambiguë de manière à trouver au moins une zone ambiguë. Plus précisément, nous modifions les seuils d_1 de 3 à 0 et d_r de 0 à 2 par pas de 0,1 jusqu'à ce que nous trouvions au moins une zone ambiguë. Le seuil d_1 élevé et le seuil d_r peu élevé nous permettent de trouver plus de zones ambiguës. Pour plus de détails sur ces seuils, nous renvoyons le lecteur à [Su *et al.* 2009].
- Lorsque les zones ambiguës sont trouvées, nous extrayons des *strokes* primaires. Puis, nous appliquons la méthode de regroupement de *strokes*

primaires présentée dans la section 3.3.2. Nous essayons de découper le *stroke* en au moins 2 sous-*strokes* en modifiant les paramètres de la méthode de regroupement. Plus précisément, nous modifions le paramètre w_θ de 1 à 0 et γ_θ de 0 à 0,5 par pas de 0,1, jusqu'à ce que le *stroke* soit découpé en au moins 2 sous-*strokes*. Le paramètre w_θ peu élevé et le paramètre γ_θ élevé nous permettent de regrouper moins de *strokes* primaires (donc, nous permettent de découper le *stroke* en plus de sous-*strokes*). Pour plus de détail sur ces seuils, nous renvoyons le lecteur à la section 3.3.2.

La Figure 5.15 montre les *strokes* dans un *cluster* d'un invariant avant le découpage et la Figure 5.16 montre le résultat du découpage.

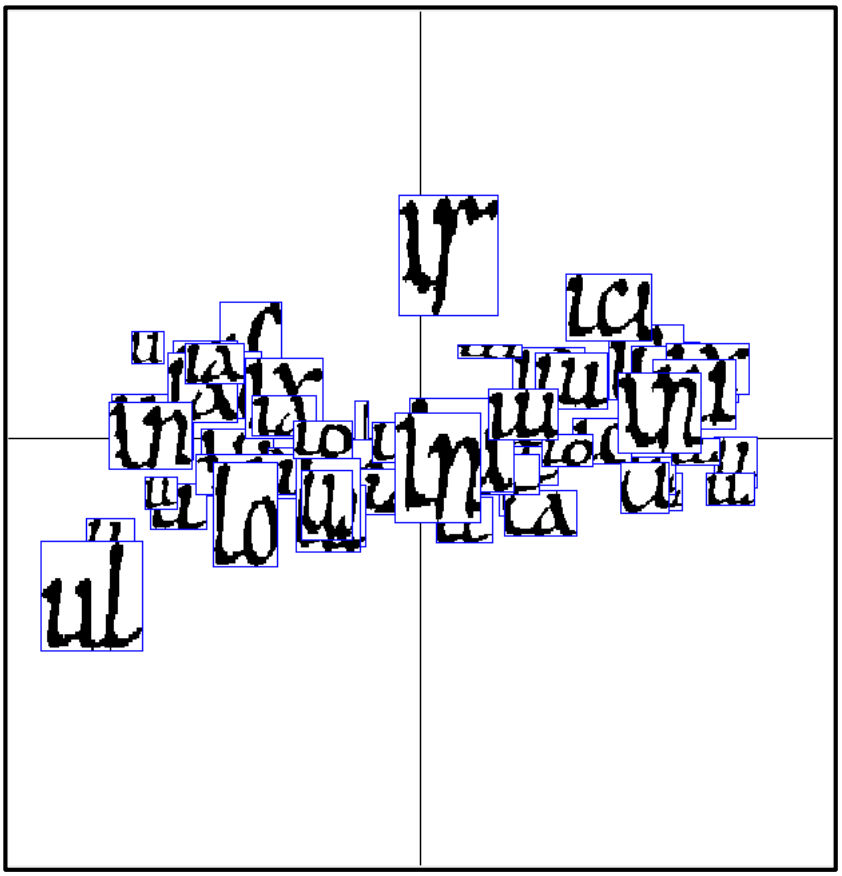


Figure 5.15 : Un *cluster* de *strokes* avant le découpage

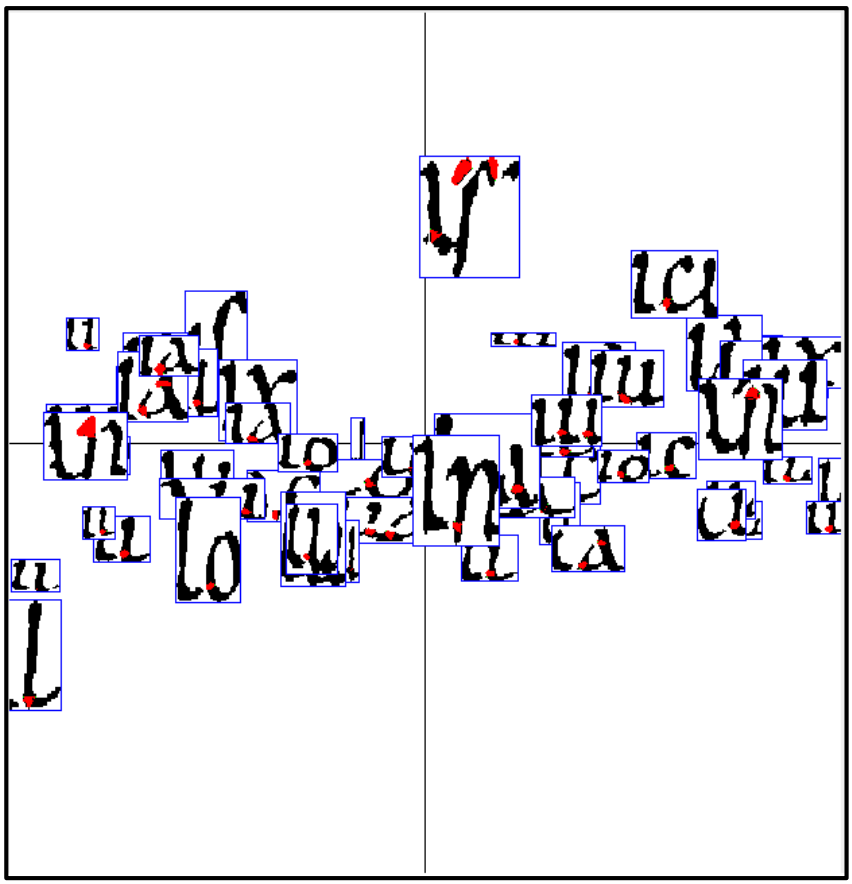


Figure 5.16 : Un résultat du découpage de *strokes*

Nous pouvons constater que notre méthode de découpage de *strokes* donne, visuellement un résultat généralement satisfaisant. Comme précédemment, les modifications engendrées par les retours de l'utilisateur (ici les modifications de paramétrages) restent cantonnés aux *strokes* concernés (*strokes* du *cluster* à découper).

Il reste quelques mauvais résultat comme :

Ces mauvais résultats sont liés à un jeu de paramètres qui est difficile à régler. Dans l'avenir, nous allons chercher différentes méthodes pour améliorer ce résultat. Par exemple, une méthode qui permette à l'utilisateur d'indiquer (ou de valider / invalider) les points de découpage sur quelques *strokes*. Grâce à ces informations données par l'utilisateur, le système fait un apprentissage sur les paramètres. À partir de la connaissance apprise, le système fait un découpage automatique de *strokes* en modifiant automatiquement ses paramètres.

5.2.2.2. Regroupement de strokes

Cette opération permet à l'utilisateur de regrouper dans l'espace spatial plusieurs *strokes* en un nouveau *stroke*. La Figure 5.17 montre l'interface qui permet à l'utilisateur de faire le regroupement des *strokes* :

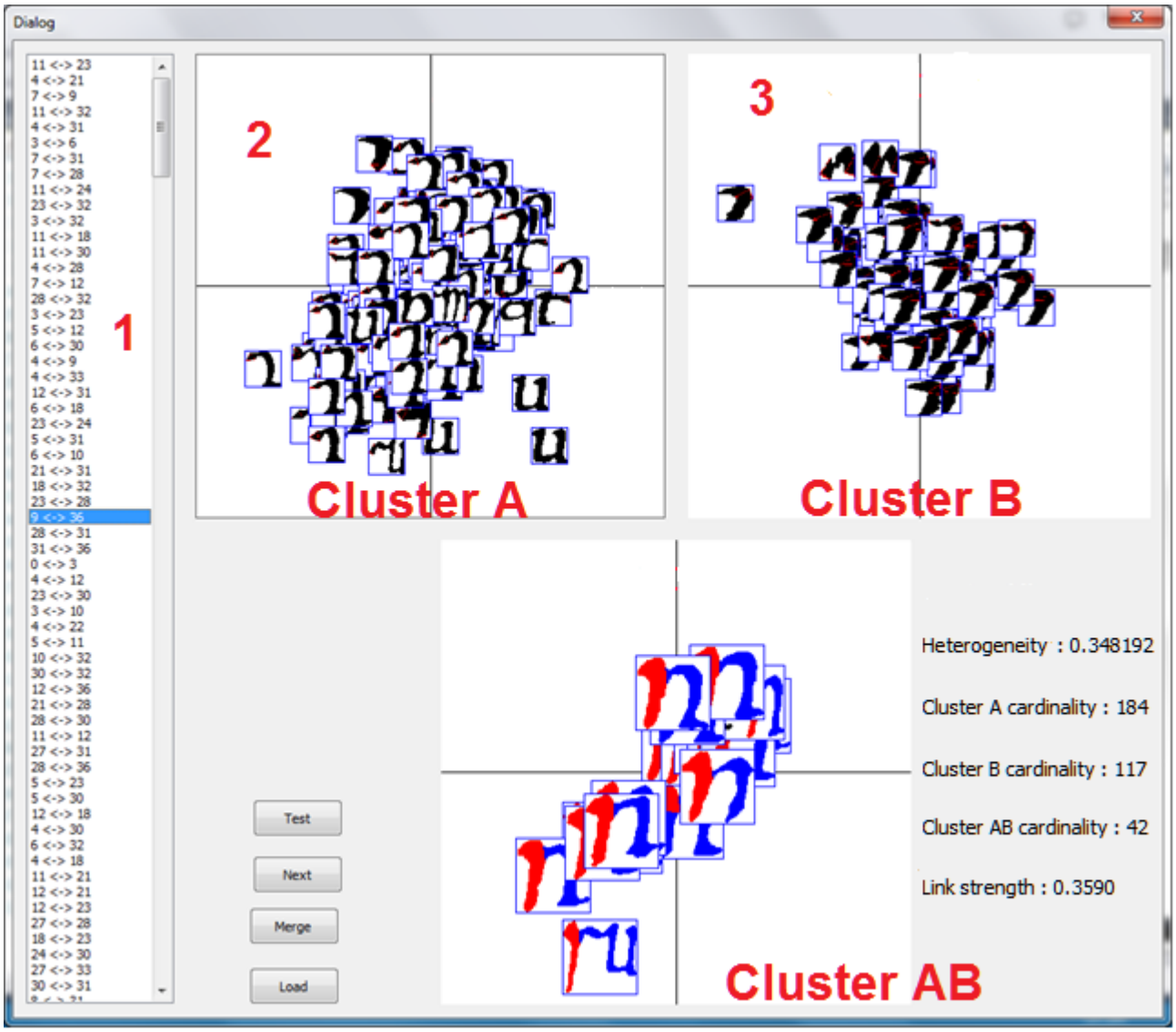


Figure 5.17 : L'interface de regroupement de *strokes* dans l'espace spatial

Dans cette interface, la fenêtre 1 donne une liste des paires de *clusters* dont les *strokes* sont connectés (spatialement). Ces paires de *clusters* sont ordonnées par le nombre de connexions de *strokes* entre les 2 *clusters*. Supposons que *cluster* A a des *strokes* connectés (spatialement) aux *strokes* de *cluster* B. La fenêtre 2 présente

les *strokes* du *cluster* A et la fenêtre 3 présente les *strokes* du *cluster* B. La fenêtre 4 présente les images des composantes connexes dans chacune desquelles, nous trouvons un *stroke* du *cluster* A (présenté en bleu) connecté à un *stroke* du *cluster* B (présenté en rouge). Supposons qu'un *stroke* du *cluster* A est regroupé avec un *stroke* du *cluster* B et forme un nouveau *stroke*. Nous appelons le groupe de ces nouveaux *strokes* : le *cluster* AB.

Pour procéder au regroupement, nous prenons également en compte l'agencement spatial des *strokes* dans la composante connexe qui les contient. Cet agencement spatial est codé par 4 codages : 1 – haut ; 2 – bas ; 3 – gauche ; 4 – droite (voir la Figure 5.18). Dans une paire de *clusters* dont les *strokes* sont connectés (spatialement), nous faisons seulement le regroupement des *strokes* dont l'agencement spatial le plus fréquent.

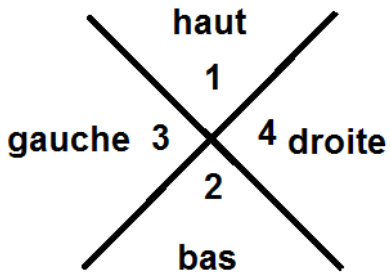


Figure 5.18 : Codage de la relation spatiale entre les *strokes*

Après le regroupement, un nouveau *cluster* qui contient les *strokes* regroupés est produit. Les *clusters* A et B sont mis à jour, avec suppression dans ces clusters de *strokes* qui appartiennent maintenant au *cluster* AB.

Pour aider l'utilisateur dans la décision du regroupement des *strokes* du *cluster* A connectés aux *strokes* du *cluster* B, nous présentons à l'utilisateur dans notre interface les mesures suivantes :

- La cardinalité du *cluster* A ($|A|$) : le nombre de *strokes* dans le *cluster* A
- La cardinalité du *cluster* B ($|B|$) : le nombre de *strokes* dans le *cluster* B
- La cardinalité du *cluster* AB ($|AB|$). Elle est aussi le nombre de connexions (dans l'agencement spatial le plus fréquent) entre les *strokes* du *cluster* A et les *strokes* du *cluster* B.
- L'intensité de la connexion entre les clusters A et B : $IC(A,B) = \max\left(\frac{|A|}{|AB|}, \frac{|B|}{|AB|}\right)$

- La mesure de d'hétérogénéité du *cluster* AB : $H(AB)$

L'utilisateur peut consulter ces propriétés pour décider du regroupement ou non des *strokes*. Ces propriétés peuvent également être utilisées pour décider automatiquement sans validation de l'utilisateur du regroupement des *strokes* (voir ci-après).

La mesure d'hétérogénéité du *cluster* AB est calculée par la formule suivante :

$$H(AB) = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} dist(s_i, s_j)$$

Où N est la cardinalité du *cluster* AB, s_i et s_j sont les *strokes* dans le *cluster* AB. $dist(s_i, s_j)$ est la distance euclidienne entre le *stroke* s_i et le *stroke* s_j. Pour calculer cette distance entre le *stroke* s_i et le *stroke* s_j, nous comparons les vecteurs de caractéristiques des *strokes*, présentés dans la section 3.4.2.

En parallèle avec ce regroupement semi-automatique, nous proposons une méthode de regroupement automatique de *strokes*. Notre méthode est la suivante :

1) Pour chaque paire de *clusters* A et B, calculer les propriétés suivantes :
   - La cardinalité du *cluster* regroupé AB : $|AB|$
   - La cardinalité du *cluster* A : $|A|$
   - La cardinalité du *cluster* B : $|B|$
   - L'intensité de la connexion entre les clusters A et B : $IC(A,B) = \max\left(\frac{|A|}{|AB|}, \frac{|B|}{|AB|}\right)$
   - La mesure d'hétérogénéité du *cluster* AB : $H(AB)$
2) Si deux conditions suivantes sont satisfaites, alors nous faisons le regroupement (de la même manière que précédemment) :
   - La condition d'homogénéité du *cluster* AB : $H(AB) < \delta$. δ est un seuil prédéfini. Nous fixons heuristiquement dans notre application $\delta = 0{,}3$.
   - La condition de la connexion entre les clusters A et B : $IL(A,B) > \mu$. μ est un seuil prédéfini. Nous fixons heuristiquement dans notre application $\mu = 0{,}2$.

3) Faire une nouvelle itération avec les nouveaux *clusters,* jusqu'à ce qu'il n'y ait pas de changement.

Dans notre méthode, deux conditions sont nécessaires : la condition de l'homogénéité du *cluster* AB et la condition de la cardinalité du *cluster* AB doivent être simultanément satisfaites. Nous donnons 2 exemples dans lesquels les deux conditions ne sont pas simultanément satisfaites et le système ne fait pas le regroupement.

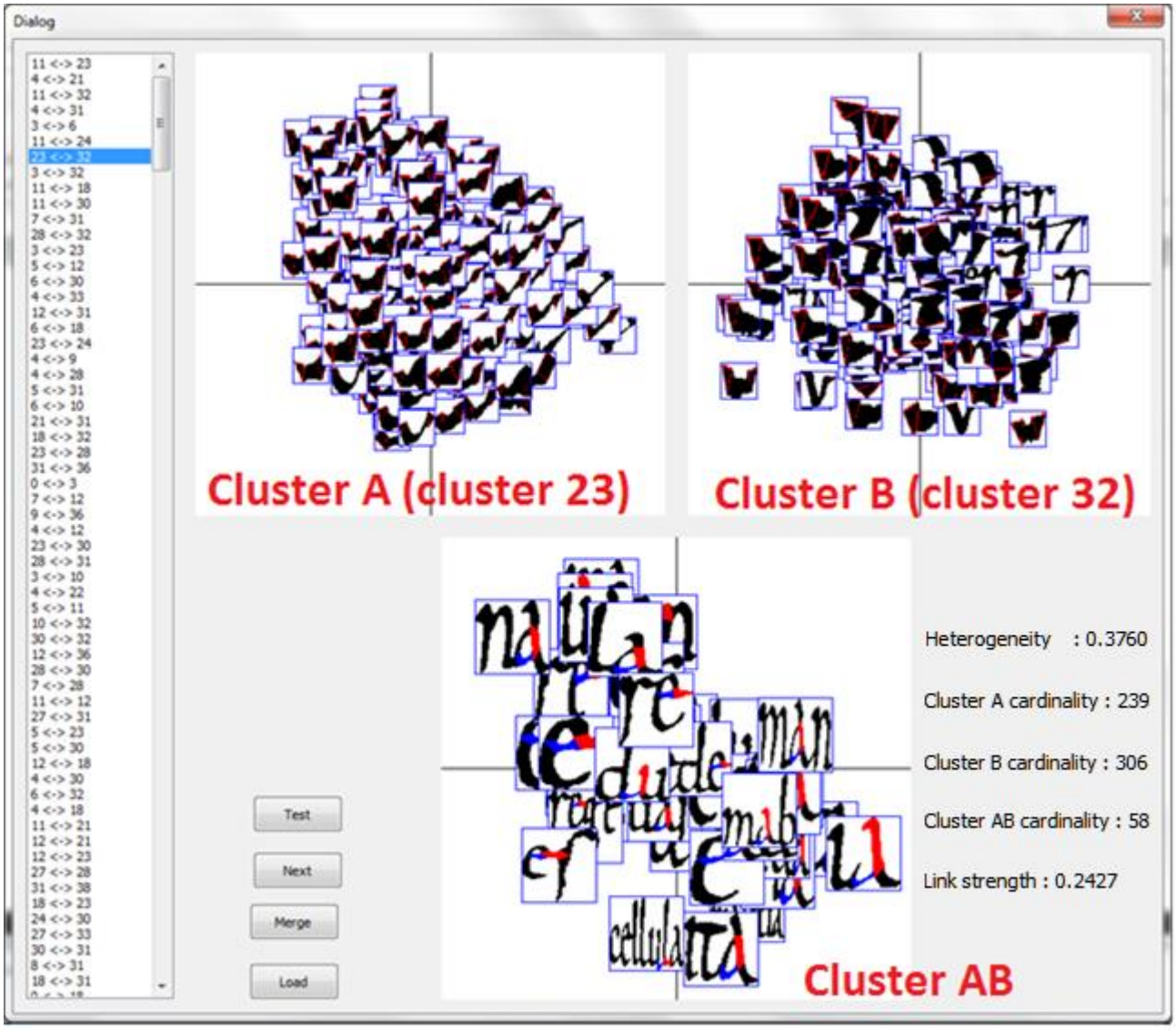


Figure 5.19 : Un exemple : le *cluster* AB satisfait la deuxième condition (la condition de la cardinalité du *cluster*) mais ne satisfait pas la première condition (la condition de l'homogénéité).

La Figure 5.19 montre un exemple où les *strokes* du *cluster* 23 connectent avec les *strokes* du *cluster* 32. Nous voyons que dans le *cluster* AB qui résulterait du

regroupement, la variété des *strokes* est élevée. Donc, il ne faut pas regrouper ces deux *clusters*. Le *cluster* AB des *strokes* regroupés satisfait la deuxième condition (la condition de la cardinalité du *cluster* : $IC(A,B) = \max\left(\frac{58}{306}, \frac{58}{239}\right) = 0{,}242 > 0{,}2$), mais il ne satisfait pas la première condition (la mesure d'hétérogénéité : $H(AB) = 0{,}376 > 0{,}3$). Par conséquence, le système ne regroupe pas ces deux *clusters*.

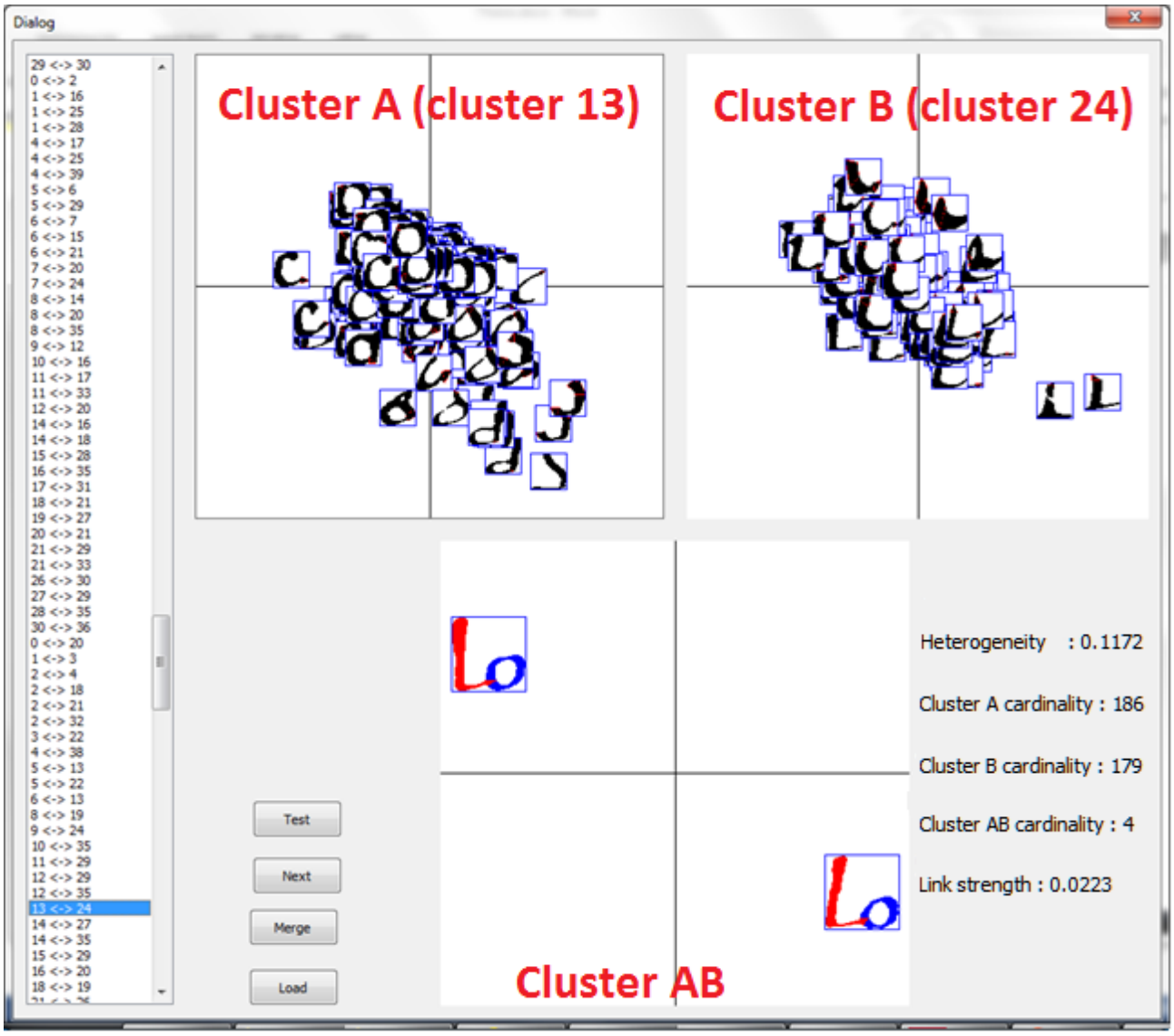


Figure 5.20 : Un exemple : le *cluster* AB satisfait la première condition (la condition de l'homogénéité) mais ne satisfait pas la deuxième condition (la condition de la cardinalité du *cluster*).

La Figure 5.20 montre un exemple où les *strokes* du *cluster* 13 sont connectés avec les *strokes* du *cluster* 24. Nous voyons que, dans le *cluster* des *strokes* regroupés, les *strokes* sont très homogènes. Mais, la quantité du *cluster* des *strokes* regroupés

est faible. Dans cet exemple, la fréquence de la forme « lo » est faible, donc, il n'y a pas d'intérêt de regrouper le *cluster* 13 et le *cluster* 24. Le *cluster* AB des *strokes* regroupés satisfait la première condition (la mesure d'hétérogénéité : $H(AB) = 0{,}117 < 0{,}3$), mais ne satisfait pas la deuxième condition ($IC(A,B) = \max\left(\frac{4}{186}, \frac{4}{179}\right) = 0{,}022 < 0{,}2$). Par conséquence, le système ne regroupe pas ces deux *clusters*.

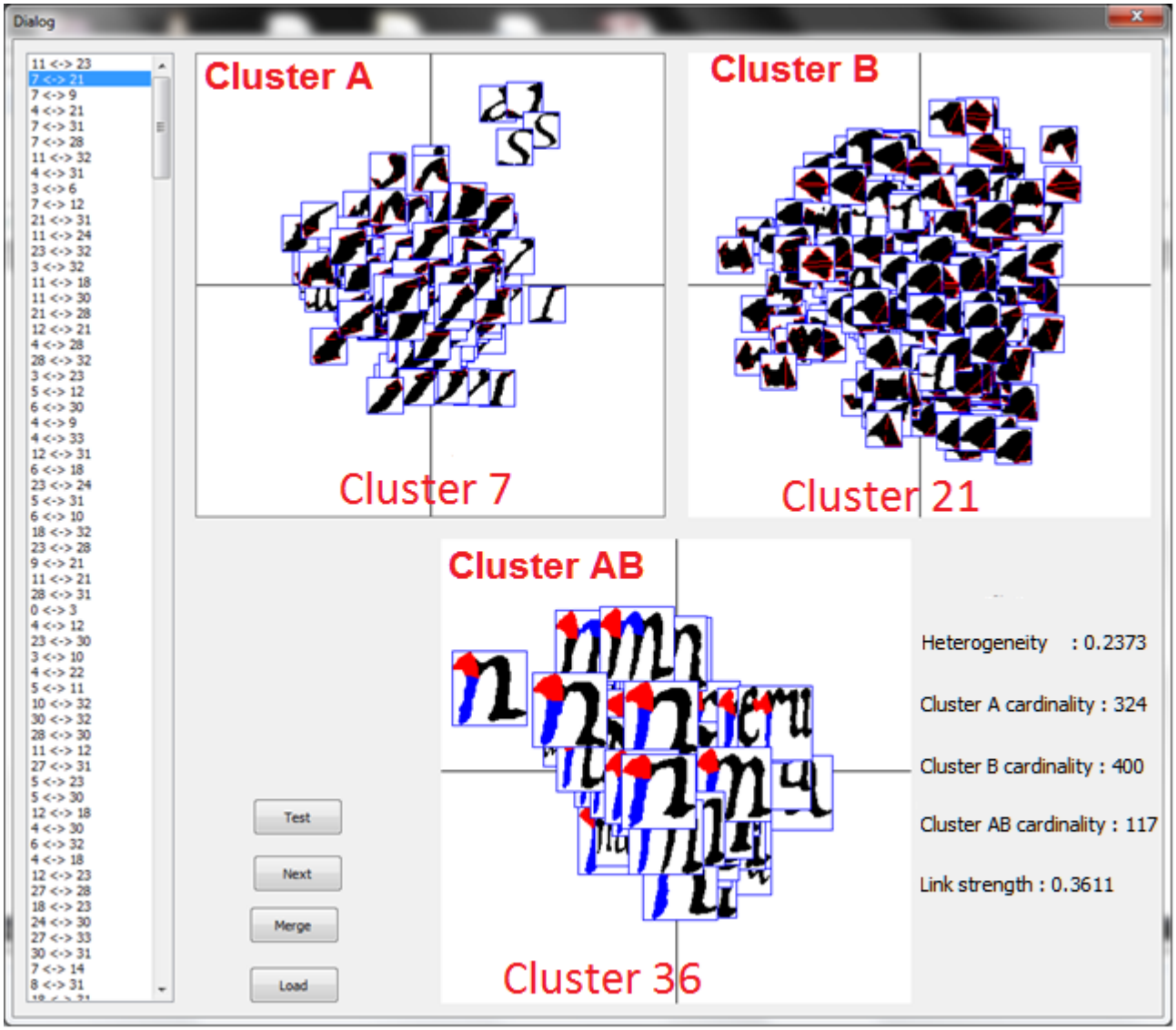


Figure 5.21 : Un exemple : le regroupement de *strokes* de 2 *clusters* satisfait 2 conditions de regroupement automatique. Les *strokes* de ces 2 *clusters* sont regroupés

En utilisant ce mécanisme de regroupement automatique, le système produit les invariants qui peuvent avoir un intérêt pour la composition de requêtes par un humain. Par exemple, la Figure 5.21 montre le cas où les *strokes* du *cluster* 7 sont

connectés avec les *strokes* du *cluster* 21. La mesure de compacité des *strokes* regroupés est 0,237 < 0,3. La cardinalité du *cluster* des *strokes* regroupés satisfait la deuxième condition de regroupement : $\max\left(\frac{117}{400}, \frac{117}{324}\right) = 0{,}361 > 0{,}2$. Donc, nous regroupons les *cluster* 7 et *cluster* 21. Un nouveau *cluster*, *cluster* 36 est produit.

Anote que, par le même procédé, le système regrouperait automatiquement les clusters 36 et 9 (voir la Figure 5.22).

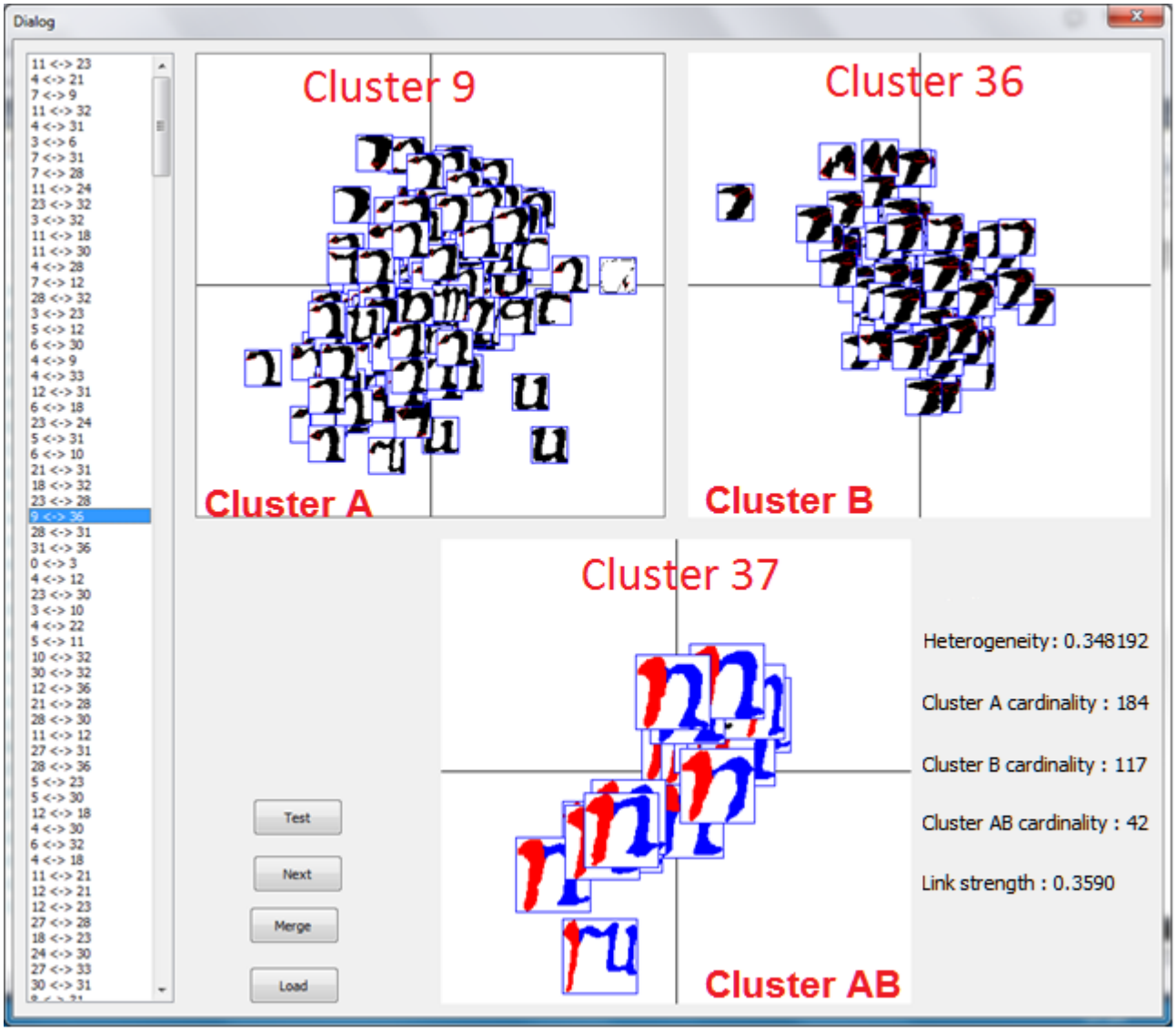


Figure 5.22 : Un exemple : le regroupement de *strokes* de 2 *clusters* satisfait 2 conditions de regroupement automatique. Les *strokes* de ces 2 *clusters* sont regroupés

Dans cette section, nous avons présenté la méthode de regroupement semi-automatique et automatique de *strokes* dans l'espace spatial, avec quelques

exemples pour prouver la cohérence de cette méthode. Ici, nous donnons un exemple qui montre les invariants avant et après le regroupement :

La Figure 5.23 montre les invariants extraits de la base « Saint Gall ». Les invariants sont extraits en utilisant la méthode de regroupement de *strokes* de niveau 1 (voir la section 4.2.3.2).

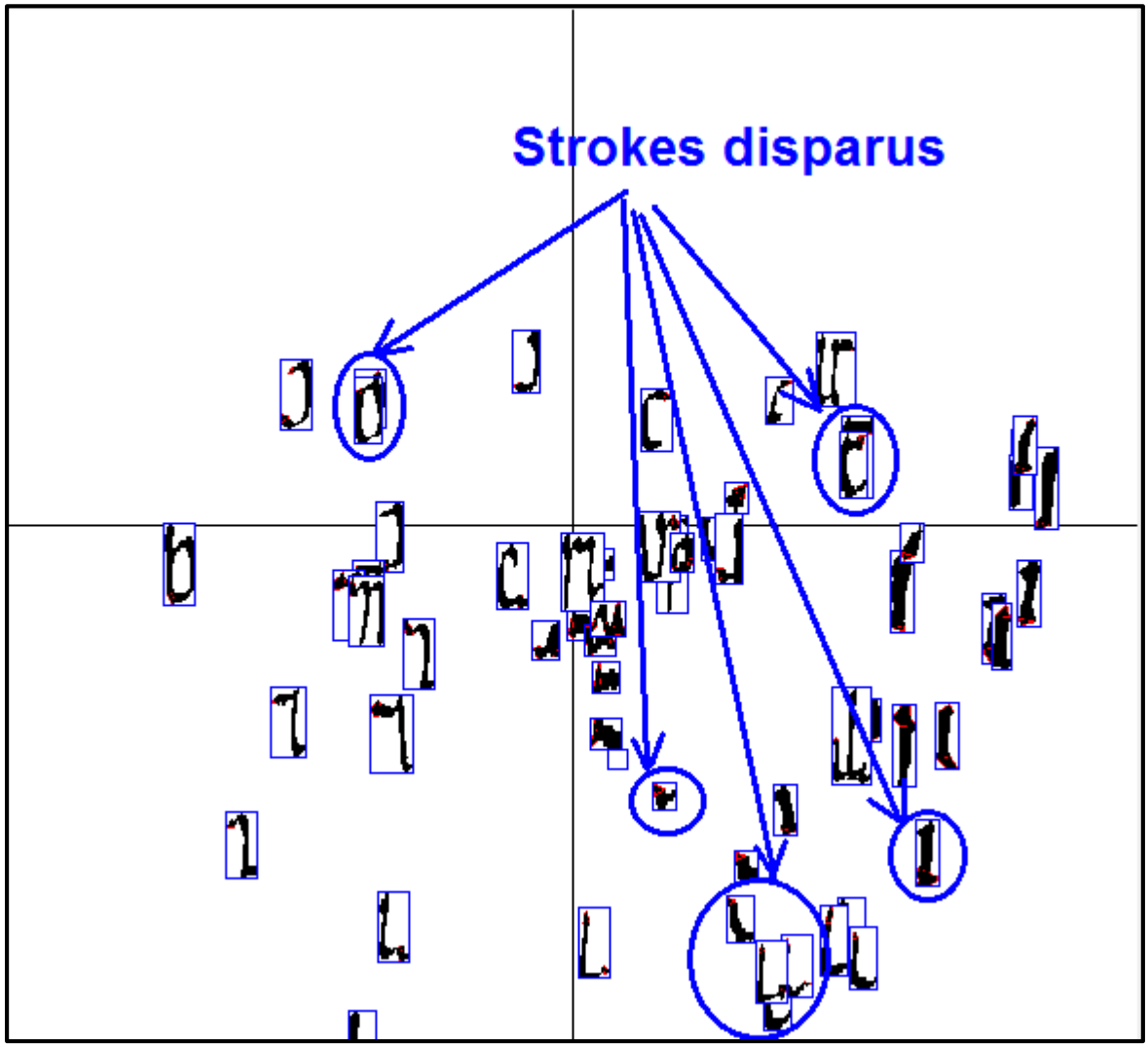


Figure 5.23 : Invariants extraits de la base « Saint Gall » avant le regroupement

La Figure 5.24 montre les invariants résultants du raffinement automatique de *strokes* dans l'espace spatial après 10 itérations (10 opérations de regroupement).

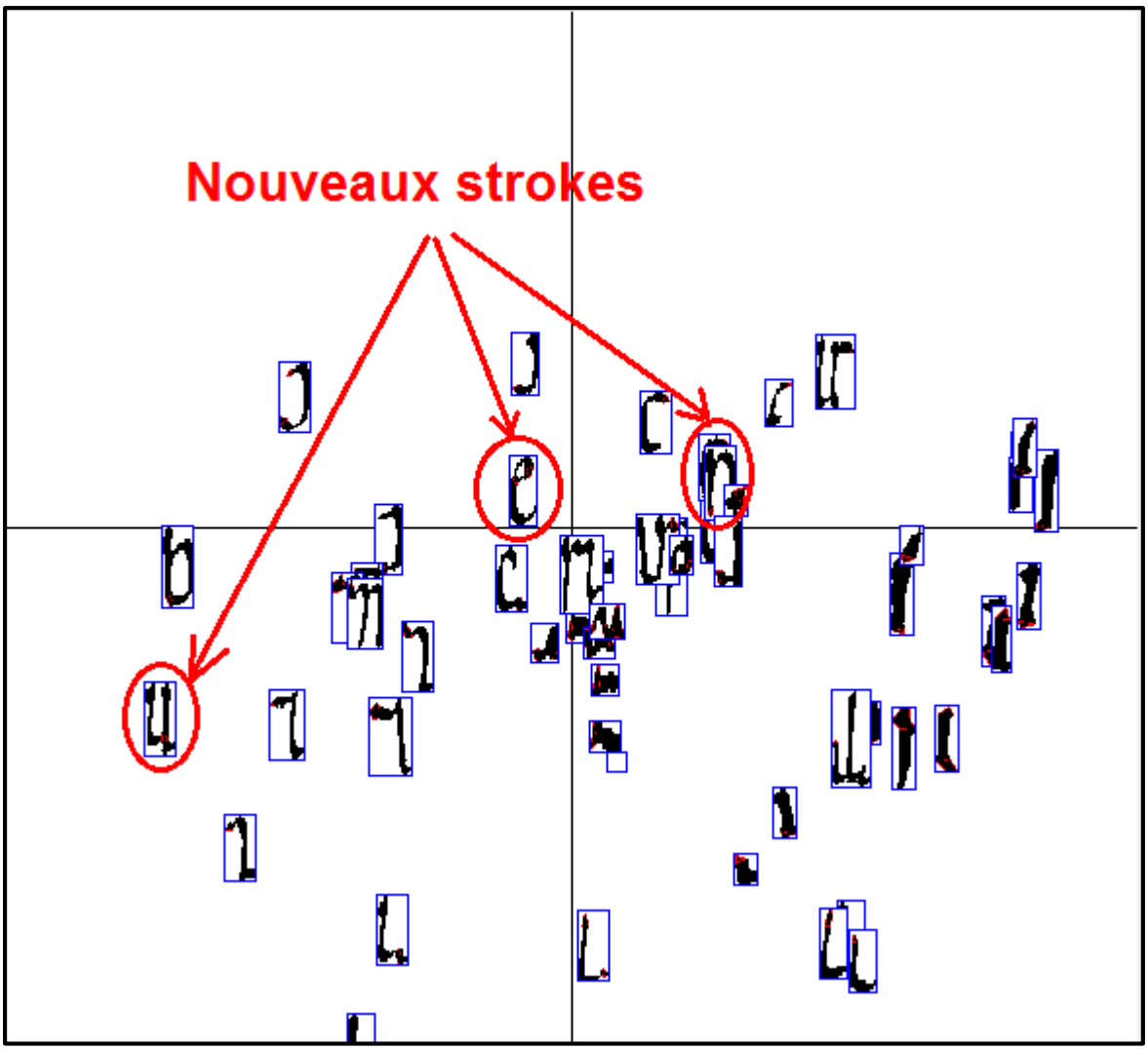


Figure 5.24 : Invariants résultant du raffinement automatique dans l'espace spatial après 10 itérations

Nous pouvons constater que notre méthode de regroupement automatique donne plus ou moins de bons résultats. Quelques *strokes* qui peuvent avoir un intérêt pour la composition de requêtes par un humain comme :

 , sont produits après le raffinement. En rajoutant un seuil sur la cardinalité des clusters initiaux A et B après le regroupement, certains *strokes* qui nous ne semblent pas avoir d'intérêt pour la composition de requêtes par un humain comme :

 , auraient disparu après le raffinement.

5.2.2.3. Discussion

Nous avons présenté dans la section 5.2.2 la méthode de raffinement interactif dans l'espace spatial. L'objectif de cette méthode est de modifier spatialement les *strokes* en se basant sur des interactions de l'utilisateur, pour qu'il puisse obtenir des invariants qui peuvent avoir un intérêt pour la composition de requêtes par un humain. Généralement, la méthode de raffinement dans l'espace spatial que nous proposons donne de bons résultats.

Les invariants produits après le raffinement dans l'espace spatial ne peuvent pas être évalués par notre méthode d'évaluation d'invariants (voir section 4.3) parce que nous devons segmenter les images dans la base de test en *strokes* et pour chaque *stroke* segmenté, nous devons retrouver l'invariant le plus similaire à ce *stroke*. Pour qu'il puisse retrouver un invariant similaire à un *stroke* segmenté, nous utilisons la même méthode d'extraction de *strokes* que nous utilisons pour extraire des invariants. Mais, si les invariants sont modifiés spatialement, nous ne pouvons trouver aucun invariant similaire aux *strokes* segmentés. Donc, la méthode d'évaluation d'invariants que nous proposons ne peut pas évaluer les invariants s'ils sont modifiés spatialement.

5.3. Conclusion

Nous avons présenté dans ce chapitre, une vue globale sur la composition interactive de requêtes pour la recherche de mots. Lorsque la composition de requête se fait à partir de l'ensemble des invariants extraits, nous proposons dans ce chapitre la méthode de raffinements interactifs des invariants pour que les invariants aient un intérêt pour la composition de requêtes.

Nous avons présenté dans la section 5.2.1 la méthode de raffinement interactif dans l'espace de caractéristiques. L'objectif de cette méthode est de modifier les *clusters* de *strokes* avec l'aide de l'utilisateur pour obtenir des invariants plus représentatifs. Nous avons conclu que la qualité des invariants après le raffinement est améliorée. Néanmoins, le temps total pour faire un raffinement est assez long. Ce phénomène est principalement causé par les opérations sur le disque dur (récupérer les caractéristiques de *strokes*, récupérer la structure des clusters, enregistrer le résultat du raffinement, *etc.*). Dans l'avenir, nous allons chercher à améliorer l'ergonomie de l'interface homme-machine que nous avons développée pour le raffinement dans l'espace de caractéristiques et réduire le temps des opérations sur le disque dur.

Nous avons présenté dans la section 5.2.2 la méthode de raffinement interactif dans l'espace spatial. L'objectif de cette méthode est de modifier spatialement les *strokes* en se basant sur des interactions de l'utilisateur, pour que l'utilisateur puisse obtenir des invariants qui peuvent avoir un intérêt pour la composition de requêtes par un humain. Généralement, la méthode de raffinement dans l'espace spatial que nous proposons donne de bons résultats.

Le module de la composition interactive de requêtes n'est pas achevé, parce que, nous n'avons pas encore développé l'interface à l'utilisateur qui lui permet de composer sa requête à partir des invariants extraits et de mesurer le véritable intérêt pour l'utilisateur de cette composition. Dans l'avenir, nous allons compléter ce module en construisant une interface intuitive et conviviale pour cette composition.

Chapitre 6 : Conclusion et perspectives

Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.

Albert Einstein

L'objectif de cette thèse est de proposer un système générique, omni-langage et interactif de recherche de mots dans des collections de documents (anciens ou modernes, manuscrits ou imprimés), fonctionnant même si le langage n'est pas connu. Nous nous plaçons dans le contexte où le contenu du document est homogène (même scripteur, d'écriture soignée). Pour atteindre cet objectif, nous faisons un état de l'art des systèmes de recherche de mots existants dans la littérature que nous avons présentés dans le chapitre 2, ainsi que leurs avantages et leurs inconvénients. En nous inspirant des méthodes de la littérature, nous proposons un système de recherche de mots basé sur des invariants qui est conforme aux objectifs poursuivis.

Le point clé du système que nous proposons est les ***invariants***. Nous définissons les invariants comme les formes les plus fréquentes dans la collection de documents. Pour la recherche des mots, les invariants peuvent servir à construire des signatures structurelles pour représenter les images de mots. Pour le requêtage, l'utilisateur pourra formuler sa requête en utilisant ces invariants, grâce à une interface visuelle ou au codage Unicode des invariants (composition de requêtes par chaîne de caractères tapés au clavier).

Cela lui permettra de se soustraire à la phase préalable de recherche du mot à retrouver dans la collection de documents, phase couteuse en temps mais nécessaire pour la plupart des systèmes de recherche de mots existants. Nous avons présenté, dans le chapitre 3, la méthode d'extraction d'invariants, basée sur l'extraction de *strokes* et le *clustering* de *strokes*. Les *strokes* sont des parties de l'écriture qui apparaissent fréquemment dans la collection de documents. Pour extraire les invariants, il faut extraire les *strokes* et les regrouper en *clusters* (par une méthode de *clustering*). Les représentants des *clusters* de *strokes* sont les invariants que nous devons extraire. Nous avons présenté dans les sections 3.2 et 3.3 la méthode d'extraction de *strokes* et dans la section 3.4, la méthode de *clustering* de *strokes* pour extraire des invariants.

Lorsque les invariants sont extraits, il est nécessaire d'avoir une évaluation pour mesurer la qualité des invariants. Nous ne pouvons pas mesurer directement la qualité sémantique des invariants, à cause de la difficulté dans l'obtention d'une vérité terrain qui permette de juger de la sémantique des invariants extraits. Par contre, nous avons proposé 2 mesures quantitatives qui mesurent 2 caractéristiques des invariants : l'exhaustivité et l'unicité. Nous avons présenté dans la section 4.3 une mesure de qualité d'invariants basée sur ces 2 caractéristiques. Nous avons montré dans des expérimentations la cohérence de cette évaluation, notamment avec l'évaluation, du système de recherche de mots présenté en section 2.

Nous avons présenté dans la section 4.2 le système de recherche de mots que nous proposons, qui utilise les invariants extraits pour construire des signatures structurelles des mots. La signature de l'image de mot est un graphe dont les nœuds sont les invariants représentant au mieux le mot, et les arêtes sont les relations spatiales entre ces invariants. Grâce à cette représentation, la dissimilarité entre les images de mots est calculée comme une distance d'édition approximative des graphes correspondants. Suite aux expérimentations présentées dans la section 4.2.4, nous avons conclu que notre système peut s'adapter aux différents types de documents (quel que soit leur script/langage, leur ancienneté, leur mode d'écriture manuscrit ou imprimé) sans connaissance a priori concernant document recherché mais au mix d'un paramétrage adapté. Ce paramétrage pourrait se faire en fonction du type de script utilisé (logo graphique, basé sur un alphabet, *etc.*) sans nécessairement réaliser d'apprentissage supervisé avec des mots du langage spécifiquement utilisé (seulement avec des exemples d'apprentissage de son type). Notre système de recherche donne meilleur résultat avec les documents homogènes que les documents non-homogènes. Un inconvénient de notre système de recherche est qu'il ne permet pas de retrouver une portion d'un « mot » et qu'il dépend fortement de la segmentation explicite du mot en *strokes*. C'est la raison pour laquelle notre système n'est pas applicable pour des langages où les mots ne sont pas séparés par des espaces comme par exemple le cambodgien. Pour contourner ce problème, dans l'avenir, nous allons étudier une autre mesure de dissimilarité qui nous permet de retrouver une portion d'un « mot », par exemple la distance d'édition de graphes basée sur la mise en correspondance des sous-graphes [Raveaux et al. 2010; Bunke & Shearer 1998].

Pour que l'utilisateur puisse utiliser les invariants pour formuler l'image de requête, les invariants doivent faire sens. C'est-à-dire, ils doivent être compréhensibles pour un humain qui connaît suffisamment le script et le langage utilisé pour juger leur pertinence, et en nombre suffisamment restreint pour que leur utilisation pour la composition de requêtes puisse se faire de manière

ergonomique. Or, les invariants retournés par la machine ne sont pas nécessairement ceux attendus par l'humain en raison du fossé sémantique existant entre la représentation basée sur des indices de bas niveau sémantique calculée automatiquement par la machine, et la perception plus sémantique de l'utilisateur. Pour contourner ce problème, nous proposons une méthode de raffinements interactifs des invariants, présentée dans la section 5.2, qui consiste en 2 parties : le raffinement dans l'espace de caractéristiques et le raffinement dans l'espace spatial.

Le raffinement dans l'espace de caractéristiques (présenté dans la section 5.2.1) a pour objectif de modifier les clusters de *strokes* avec l'aide de l'utilisateur pour obtenir des invariants plus représentatifs. Nous avons calculé la mesure de qualité des invariants avant et après le raffinement et nous avons conclu que la qualité des invariants après le raffinement est améliorée. Par contre, le temps total pour faire un raffinement est assez long. Ce phénomène n'est pas causé par le calcul dans l'espace de caractéristiques mais par les opérations sur le disque dur (récupérer les caractéristiques de *strokes*, récupérer la structure des clusters, enregistrer le résultat du raffinement, *etc.*) Le nombre de clics à réaliser pour une opération donnée est aussi important. Dans l'avenir, nous allons chercher à améliorer l'ergonomie de l'interface homme-machine que nous avons développée et réduire le temps des opérations sur le disque dur. Le raffinement dans l'espace spatial (présenté dans la section 5.2.2) a pour objectif de modifier spatialement les *strokes* en se basant sur des interactions de l'utilisateur, pour que l'utilisateur puisse obtenir des invariants qui peuvent avoir un intérêt pour la composition de requêtes par un humain. Généralement, la méthode de raffinement dans l'espace spatial que nous proposons donne de bons résultats.

Pour la composition de requêtes, la méthode de raffinement interactifs des invariants que nous proposons a pour objectif de produire des invariants significatifs pour l'humain. Il nous faudra donc travailler à une interface qui permette à l'utilisateur de composer interactivement sa requête à partir des invariants extraits. Dans l'avenir, nous allons compléter la méthode de composition de requêtes en construisant une interface intuitive et conviviale.

Annexes

Annexe A. Les descripteurs de formes

Pour que les *strokes* soient groupés en *clusters*, les caractéristiques de *strokes* doivent être extraites. Les caractéristiques de *strokes* sont les représentations numériques qui doivent caractériser quelques propriétés essentielles comme par exemple :

- Invariance : Les *strokes* de formes similaires ont des descripteurs similaires.
- Complétude : Les descripteurs devraient être les plus complets possibles pour représenter tous les informations du *stroke*.
- Robustesse aux bruits : Les descripteurs doivent être aussi robustes que possible vis-à-vis des bruits.
- Indépendance : Deux descripteurs doivent idéalement être statistiquement indépendants. La taille du vecteur de descripteurs doit être aussi réduite que possible.

Lorsque l'on travaille avec des images binaires plutôt que des images couleurs, les descripteurs de formes sont préférés. Dans cette section, nous présentons quelques descripteurs de formes souvent utilisés :

- Les paramètres de forme :
    - Le centre de gravité
    - L'excentricité
    - Le ratio circulaire
    - La rectangularité
    - La convexité
    - La solidité
    - Les profils
- Les signatures de forme basées sur leur contour :
    - La courbure du contour
    - La représentation de l'aire
    - La représentation de l'aire triangulaire
- Les signatures de forme basées sur leur distribution spatiale :
    - Boîtes de limitation
    - Contextes de formes
- Les moments :
    - Les moments de Hu
    - Les moments de Zernike

A.1. Les paramètres de forme

Les paramètres de formes sont des caractéristiques géométriques simples représentatives de formes. Ces caractéristiques peuvent seulement discriminer grossièrement des formes très différentes. Donc, les paramètres de forme sont souvent utilisés pour du pré-filtrage, ou combinés avec des autres descripteurs pour discriminer plus finement les formes.

1. *Le centre de gravité*

Une forme est un ensemble des pixels : $\{P_i(p_{ix}, p_{iy}) | i = 1 \dots N\}$. N est le nombre de pixels dans la forme. Le centre de gravité $G(g_x, g_y)$ est calculé comme suit :

$$\begin{cases} g_x = \frac{1}{N}\sum_{i=1}^{N} p_{ix} \\ g_y = \frac{1}{N}\sum_{i=1}^{N} p_{iy} \end{cases}$$

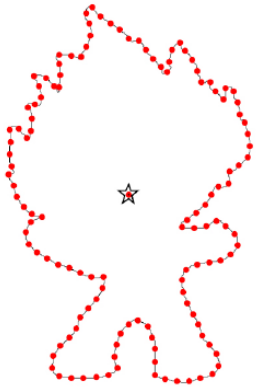


Figure 6.1 : Le centre de gravité d'une forme [Yang et al. 2008]

Le centre de gravité n'est pas invariant à l'échelle et n'est pas invariant à la rotation.

2. *La hauteur et la largeur de la forme*

La hauteur H et la largeur W de la forme sont la hauteur et la largeur de la boîte de limitation minimale de la forme.

La boîte de limitation minimale est le rectangle avec la plus petite aire dans lequel tous les pixels d'avant-plan sont situés (voir la Figure 6.2).

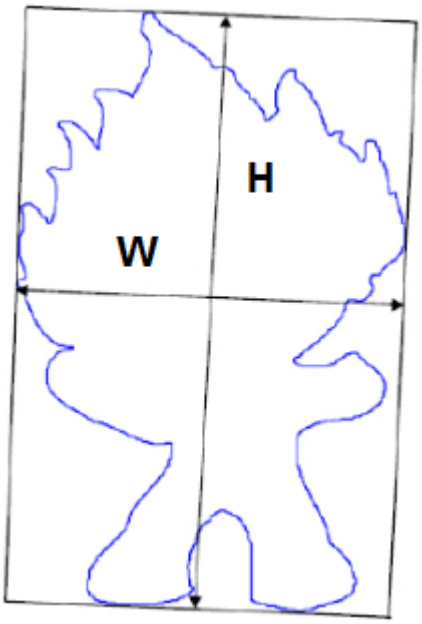


Figure 6.2 : La boîte de limitation minimale d'une forme. W et H sont les deux axes de cette boîte de limitation minimale [Yang et al. 2008]

La hauteur et la largeur de la forme ne sont pas invariantes à l'échelle et ne sont pas invariantes à la rotation.

3. L'excentricité

L'excentricité est le rapport entre la longueur de l'axe majeur et la longueur du l'axe mineur de la « boîte de limitation minimale » :

$$Ex = \frac{H}{W}$$

L'excentricité est invariante à l'échelle et à la rotation.

4. Le ratio circulaire

Le ratio circulaire représente dans quelle mesure une forme est similaire à un cercle. Il est le rapport entre l'aire de la forme A_s avec l'aire d'un cercle ayant le même périmètre A_c :

$$C = \frac{A_s}{A_c}$$

Notons le périmètre de la forme est P, on obtient :

$$C = \frac{4\pi . A_s}{P^2}$$

Le ratio circulaire est invariant à l'échelle et à la rotation.

5. La rectangularité

La rectangularité représente dans quelle mesure une forme est similaire à un rectangle . Elle est le rapport entre l'aire de la forme A_s avec l'aire de la boîte de limitation minimale de la forme A_R

$$Rect = \frac{A_s}{A_R}$$

La rectangularité est invariante à l'échelle et à la rotation.

6. La convexité

La convexité est le rapport entre le périmètre du convexe de la forme P_{convex} avec le périmètre P de la forme :

$$Convexité = \frac{P_{convex}}{P}$$

La convexité est invariante à l'échelle et à la rotation.

7. La solidité

La solidité décrit l'apparence plutôt convexe ou concave d'une forme. Elle est le rapport entre l'aire A_s de la forme et l'aire du convexe de la forme H :

$$Solidité = \frac{A_s}{H}$$

La solidité est invariante à l'échelle et à la rotation.

8. Les profils

Les profils sont les projections de la forme sur l'axe X et l'axe Y :

$$Profil_x(x_0) = \sum_{y=0}^{H_I} f(x_0, y)$$

$$Profil_y(y_0) = \sum_{x=0}^{W_I} f(x, y_0)$$

Où H_I est la longueur de l'image de la forme, W_I est la largeur de l'image de la forme, $f(x, y) = 1$ si (x, y) est un pixel de la forme. Sinon, $f(x, y) = 0$

Les profils sont invariants à la rotation. Pour calculer la similarité entre 2 formes en utilisant les profils, les tailles de ces deux formes doivent être normalisées. Les profils sont donc invariants à l'échelle.

A.2. Les signatures de la forme basées sur leur contour

Les signatures de la forme représentent la forme sous un espace d'une dimension. Ils capturent la caractéristique perceptive de la forme. Le profil de courbure du contour, la fonction de l'aire, la représentation de l'aire du triangle sont les signatures les plus souvent utilisés.

1. *Le profil de courbure*

La courbure du contour contient des informations important pour discriminer les formes. Supposons que le contour de la forme est la chaine des points : $C = (p_1, p_2, \dots, p_N)$. Pour chaque point $p_i (i = 1 \dots N)$, la courbure du contour à ce point est calculée, en utilisant une fonction d'estimation de la courbure, par exemple, la fonction d'estimation présentée dans [Plamondon & Privitera 1999]. Les courbures des points dans le contour sont donc présentés sous forme un profil.

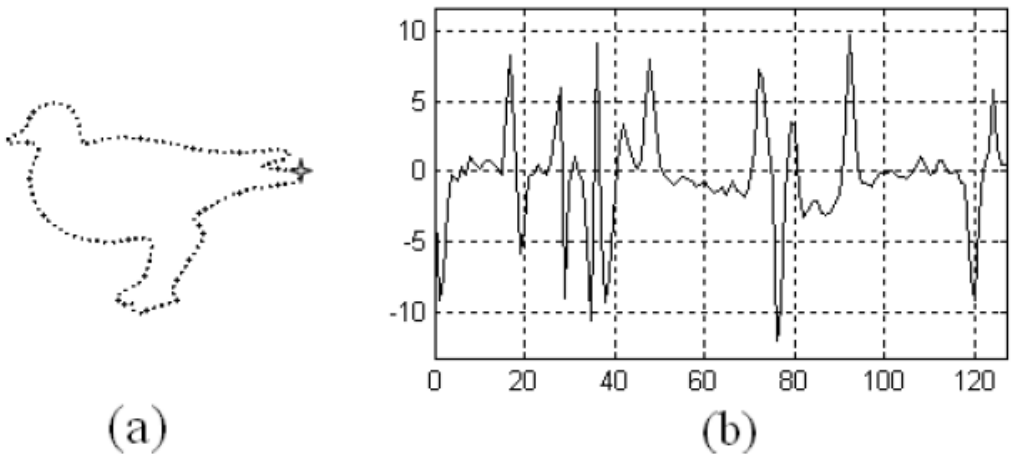


Figure 6.3 : Le profil (b) de la courbure du contour de la forme (a)

Le profil de la courbure du contour est invariant à l'échelle si nous appliquons une normalisation sur le contour. Il est aussi invariant à la rotation et à la translation. Clairement, même considéré seul, le profil de la courbure du contour peut discriminer des formes différentes.

2. La représentation de l'aire

Comme le profil de la courbure, la représentation de l'aire se base sur le contour de la forme, mais, au lieu de calculer la courbure des points dans le contour, nous calculons l'aire $S(i)$ du triangle formé entre 2 points successifs P, P_{n+1} dans le contour et le centre de gravité de la forme $G(g_x, g_y)$, comme illustré en Figure 6.4 – a.

La représentation de l'aire est ensuite présentée sous forme un profil. La Figure 6.4 montre un exemple :

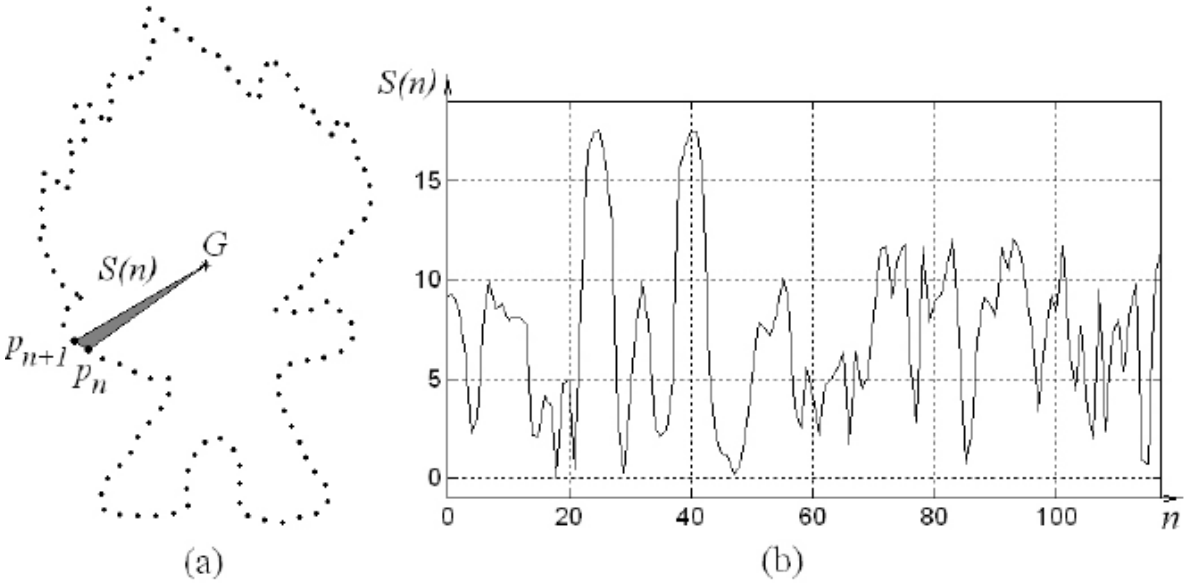


Figure 6.4 : (a) Le triangle formé par 2 points successifs dans le contour et le centre de gravité de la forme. (b) Le profil de la représentation de l'aire. [Yang et al. 2008]

La représentation de l'aire est linéaire en fonction de transformations affines. De plus, ce descripteur est invariant à la rotation et à la translation. La représentation de l'aire normalisée est invariante à l'échelle.

3. La représentation de l'aire triangulaire

La représentation de l'aire triangulaire (RAT – Représentation de l'Aire du Triangulaire) [N. *et al.* 2007] est calculée en utilisant l'aire des triangles qui sont

formés à partir des points dans le contour. Plus précisément, en suivant le contour dans le sens horaire, pour chaque triplet de points : $P_{n-t_s}(x_{n-t_s}, y_{n-t_s})$, $P_n(x_n, y_n)$, $P_{n+t_s}(x_{n+t_s}, y_{n+t_s})$ dans le contour, nous calculons la valeur $RAT(n, t_s)$ formée par ces 3 points :

$$RAT(n, t_s) = \frac{1}{2}\begin{pmatrix} x_{n-t_s} & y_{n-t_s} & 1 \\ x_n & y_n & 1 \\ x_{n+t_s} & y_{n+t_s} & 1 \end{pmatrix}$$

Où $n \in [1, N], t_s \in [1, \frac{N}{2} - 1]$, et N est le nombre de points dans le contour

Pour chaque valeur t_s différente, nous obtenons des profils $RAT(t_s)$ différents. La représentation de l'aire triangulaire est donc un descripteur multi-échelle. Elle est invariante aux transformations affines, à la rotation et à la translation.

Conclusion :

Les signatures de formes représentent une forme par un vecteur de caractéristiques, basée sur le contour de la forme. Ces descripteurs est sensible aux bruits. Donc, il n'est pas souhaitable de décrire directement la forme en utilisant les signatures de formes. Des traitements supplémentaires sont nécessaires pour augmenter sa robustesse.

A.3. Les signatures de forme basées sur leur distribution spatiale

Les signatures de forme basées sur leur distribution spatiale décrivent la forme par les relations spatiales entre leurs pixels. Quelques-uns des descripteurs les plus connus de ce type sont : « Boîtes de limitation » et « Contexte de forme ».

1. *Boîtes de limitation*

« Boîtes de limitation » sont une caractéristique proposée par [Bauckhage & Tsotsos 2005] Son idée principale est d'assimiler la forme à un ensemble de rectangles de tailles différentes. Chaque rectangle couvre une région de la forme.

**** Définition : Boîte de limitation*** : Une boîte de limitation d'un ensemble de points est le rectangle $H \times W$ (H est parallèle à l'axe O_y, W est parallèle à l'axe O_x)

de taille minimale dans lequel tous les points sont situés. Par exemple, la Figure 6.5 montre la boîte de limitation d'une forme.

La méthode de [Bauckhage & Tsotsos 2005] pour calculer la caractéristique « Boîtes de limitation » est la suivante :

- Étape 1 : Étant donné la forme S, trouver sa boîte de limitation $B(S)$.

Figure 6.5 : Une forme et sa boîte de limitation

- Étape 2 : Diviser la boîte de limitation $B(S)$ en N tranches verticales de largeurs égales S_j $(j = 1 \ldots N)$.

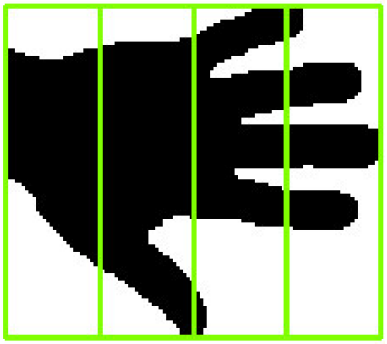


Figure 6.6 : N = 4 tranches verticales égales d'une boîte de limitation

- Étape 3 : Pour chaque tranche verticale S_j, trouver la boîte de limitation des points dans cette tranche : $B(S_j)$.

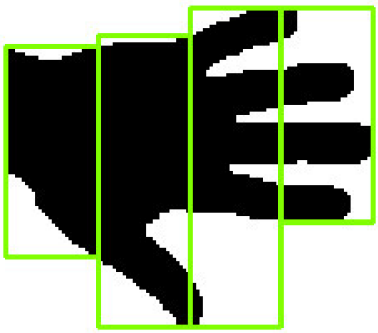


Figure 6.7 : Les boîtes de limitation des tranches verticales

- Étape 4 : Pour chaque boîte de limitation $B(S_j)$, diviser $B(S_j)$ en M tranches horizontales égales : S_{ij} $(i = 1 \dots M)$.

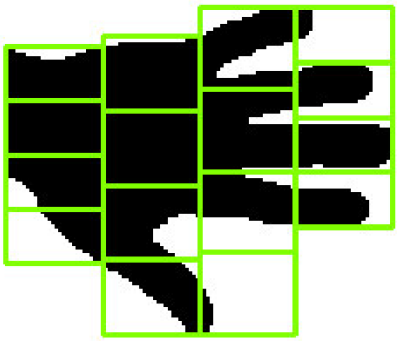


Figure 6.8 : Les tranches horizontales des boîtes de limitation verticales

- Étape 5 : Pour chaque tranche horizontale S_{ij}, trouver la boîte de limitation des points dans cette tranche : $B(S_{ij})$.

Après étape 5, la forme est approximée avec des boîtes de limitation $B(S_{ij})$.

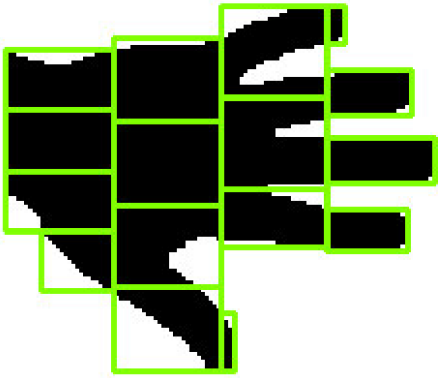


Figure 6.9 : Les boîtes de limitation au final

- Étape 6 : La forme S est finalement représentée par le vecteur de caractéristiques $\mathrm{V} = \left(\mu_x^{ij}, \mu_y^{ij}\right) i = 1 \dots M, j = 1 \dots N$

$$\mu_x^{ij} = \frac{c_x^{ij} - o_x}{w}$$

$$\mu_y^{ij} = \frac{c_y^{ij} - o_y}{h}$$

$\left(c_x^{ij}, c_y^{ij}\right)$ est la position du centre de la boîte $B(S_{ij})$.

$\left(o_x, o_y\right)$ est la position du coin supérieur gauche de la boîte $B(S)$.

$w\ et\ h$ sont la largeur et la hauteur de la boîte $B(S)$ correspondantes.

La caractéristique « boîtes de limitation » est invariante à la translation et à l'échelle. Elle est aussi robuste aux bruits. La complexité pour calculer la caractéristique « boîte de limitation » est peu élevée.

Si la méthode de normalisation présentée ci-après est utilisée, les caractéristiques « boîtes de limitation » décrive également invariante à la rotation :

- Trouver l'axe majeur de la forme. L'axe majeur de la forme est la ligne joignant les deux points P1, P2 dans le contour qui sont les plus éloignés l'un de l'autre.
- Tourner la forme pour que l'axe majeur soit parallèle à l'axe O_x et le centre de gravité de la forme soit au-dessous de l'axe majeur.

2. Contexte de forme

[Belongie *et al.* 2002] proposent la caractéristique « contexte de forme ». Cette caractéristique représente, pour chaque point dans le contour de la forme, les relations entre ce point et les autres points dans le contour. Plus précisément, pour chaque point P_i dans le contour, les auteurs calculent un histogramme de

distribution de coordonnées log-polaires relatives des points restants dans le contour h_i :

$$h_i(k) = \#\{q \neq p_i : q \in bin(k)\}$$

Où k est l'indice du bin dans l'histogramme

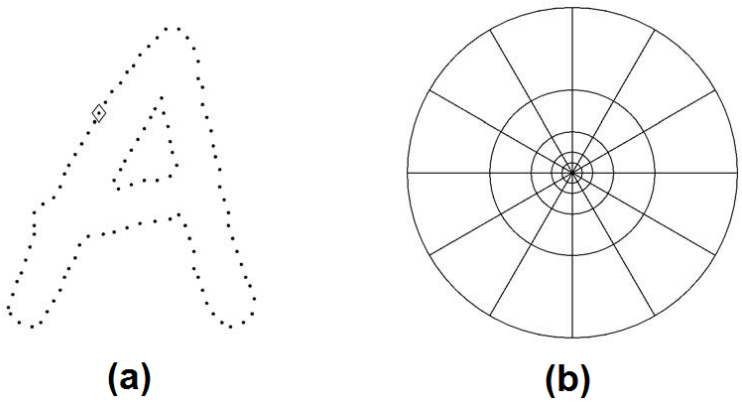


Figure 6.10 : (a) : La forme avec ses points dans le contour. (b) : L'histogramme log-polaire avec des bins utilisés dans le calcul de contexte de forme [Yang et al. 2008]

Pour que la caractéristique « contexte de forme » soit invariant à l'échelle, les auteurs de [Belongie *et al.* 2002] normalisent tous les distances par la distance moyenne α entre les paires de points dans le contour.

Pour que la caractéristique « contexte de forme » soit invariant à la rotation, dans le calcul de l'histogramme log-polaire de chaque point, au lieu de se baser sur l'axe O_x pour le calcul de l'angle, les auteurs de [Belongie *et al.* 2002] se basent sur le vecteur tangent de ce point.

Pour calculer la similarité entre deux formes, [Belongie *et al.* 2002] font une mise en correspondance (*matching*) entre deux formes et calculent le coût de ce matching. Plus précisément, les auteurs calculent le coût $C(p_i, q_j)$ de mise en correspondance d'un point p_i dans la première forme avec un point q_j dans la deuxième forme :

$$C(p_i, q_j) = \frac{1}{2}\sum_{k=1}^{K}\frac{\left(h_i(k) - h_j(k)\right)^2}{h_i(k) + h_j(k)}$$

Où K est le nombre de bins de l'histogramme log-polaire.

Les auteurs trouvent la solution optimale du matching dont le coût est le plus petit. La similarité entre les deux formes est donc le coût de la solution optimale du matching.

La caractéristique « contexte de forme » est une représentation compacte de la forme qui contient des informations riches. Elle est robuste aux bruits. De plus, elle est invariante à la rotation, à la transformation et à l'échelle. Par contre, la complexité pour calculer les contextes de forme est élevée.

A.4. Les caractéristiques de forme basées sur le calcul de moments

1. Les moments de Hu

Les moments de Hu [Celebi & Aslandogan 2005] sont les moments de la forme qui sont invariants à la translation, à l'échelle et à la rotation. Pour calculer les moments de Hu, d'abord, nous devons calculer les moments M_{pq} d'ordre (p+q) de la forme :

$$M_{ij} = \sum_x \sum_y x^p y^q I(x,y)$$

Où $I(x,y)$ est la valeur de gris du pixel (x,y) dans l'image

Puis, nous calculons les moments centrés μ_{pq} :

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y)$$

avec

$$\bar{x} = \frac{M_{10}}{M_{00}}$$

$$\bar{y} = \frac{M_{01}}{M_{00}}$$

Les moments centrés sont invariants à la translation

Ensuite, nous calculons les moments η_{pq} qui sont invariants à la translation et à l'échelle:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\left(1+\frac{p+q}{2}\right)}}$$

Finalement, les moments de Hu sont calculés par les formules suivantes :

$$I_1 = \eta_{20} + \eta_{02}$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$I_4 = (n_{30} + n_{12})^2 + (n_{21} + n_{03})^2$$

$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3n_{21} - n_{03})(n_{21} + n_{03})[3(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2]$$

$$I_6 = (n_{20} - n_{02})[(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2] + 4n_{11}(n_{30} + n_{12})(n_{21} + n_{03})$$

$$I_7 = (3n_{21} - n_{03})(n_{30} + n_{12})[(n_{30} + n_{12})^2 - 3(n_{21} + n_{03})^2] - (n_{30} - 3n_{12})(n_{21} + n_{03})[3(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2]$$

Le calcul des moments de Hu est simple. Ces moments sont invariants à la translation, à la rotation et à l'échelle. Le dernier d'entre eux (I_7) est également invariants aux transformations de type « miroir » . Par contre, ils sont sensibles aux bruits et ils ont des redondances d'information.

2. Les moments de Zernike

Les moments de Zernike [Celebi & Aslandogan 2005] sont les moments orthogonaux. Pour calculer les moments de Zernike, les coordonnées de l'image doivent être exprimées dans l'espace polaire : $(x, y) \rightarrow (r, \theta)$.

Le moment de Zernike de l'ordre n avec répétition m de la forme $I(x, y)$ est :

$$Z_{nm} = \frac{n+1}{\pi} \sum_{r} \sum_{\theta} I(r\cos\theta\,, r\sin\theta).\, R_{nm}(r).\exp(jm\theta)$$

$$R_{nm}(r) = \sum_{s=0}^{\frac{n-|m|}{2}} (-1)^s \frac{(n-s)!}{s! \times \left(\frac{n-2s+|m|}{2}\right)! \left(\frac{n-2s-|m|}{2}\right)!} r^{n-2s}$$

Les moments de Zernike sont invariants à la rotation, à la translation. Ils sont robustes aux bruits et ils portent peu de redondances de l'information. Par contre, la complexité du calcul des moments de Zernike est élevée. De plus, dans le calcul des moments de Zernike, les intégrales continues doivent être estimées par des valeurs discrètes. Cette approximation produit des erreurs dans valeurs des moments calculés.

Annexe B. Les méthodes de *clustering*

Les méthodes de *clustering* sont des méthodes statistiques d'analyse de données. Elles visent à partitionner un ensemble de données en différents *clusters* homogènes, en ce sens que les données de chaque *cluster* partagent des caractéristiques communes qui répondent le plus souvent à des critères de similarité.

Pour obtenir un bon *clustering*, il convient de :

- Minimiser l'inertie intra-classe : pour obtenir des *clusters* les plus homogènes possibles
- Maximiser l'inertie interclasse : pour obtenir des *clusters* bien différenciés

Nous pouvons définir une taxonomie des méthodes de *clustering* qui distingue deux types : les méthodes de *clustering* dures et les méthodes de *clustering* floues. Les méthodes de *clustering* dures assignent chaque objet à un et seulement un *cluster*, alors qu'avec les méthodes de *clustering* floues, un objet peut appartenir à un ou plusieurs *clusters* avec différents degrés d'appartenance. Dans notre contexte, nous considérons seulement les méthodes de *clustering* dures.

Plusieurs différents types de méthodes de *clustering* ont été proposés dans la littérature :

- ***Les méthodes par partitionnement :*** Ces méthodes partitionnent les données en se basant sur la proximité des objets dans l'espace de caractéristiques. Généralement, ces méthodes donnent une organisation non-hiérarchique des *clusters*.
- ***Les méthodes hiérarchiques :*** Ces méthodes organisent les objets dans une structure hiérarchique des *clusters*.
- ***Les méthodes basées sur la densité :*** Ces méthodes essaient de partitionner les objets en se basant sur leurs densités locales.
- ***Les méthodes basées sur des modèles :*** Ces méthodes essaient de modéliser les *clusters* et ensuite, utilisent ces modèles pour classifier les nouveaux objets.

Nous présentons dans les sections suivantes les différents types de méthodes de *clustering* citées ci-dessus. Dans notre présentation, nous utilisons les notions suivantes :

$X = \{x_i | i = 1..N\}$: L'ensemble des objets en entrée. N : le nombre d'objets

$K = \{K_j | j = 1..k\}$: L'ensemble de *clusters*. k : le nombre de *clusters*

B.1. Les méthodes par partitionnement

Les méthodes par partitionnement sont destinées à partitionner les données à k *clusters*, où k est souvent prédéfini. Quelques méthodes de ce type sont : K-moyennes, K-moyennes globales, K-médoïdes, CLARA, *etc.*

a) K-moyennes

La méthode K-moyennes [MacQueen 1967] est sans doute la plus connue des méthodes par partitionnement. Elle est une méthode itérative qui partitionne les données en k *clusters* pour que chaque point appartienne au *cluster* pour lequel la distance moyenne est la plus proche. L'idée de cette méthode est de minimiser la fonction d'objectif suivante :

$$F_{obj} = \sum_{j=1}^{k} \sum_{x_i \in K_j} \left(x_i - \mu_j\right)^2$$

Où μ_j est la moyenne des objets dans le *cluster* K_j

Plus précisément, l'algorithme K-moyenne est le suivant :

- Initialiser les k moyennes des *clusters* (par exemple, choisir au hasard 5 objets dans l'ensemble des données.
- Répéter :
  - Attribuer chaque objet x_i au *cluster* de plus proche moyenne μ_j
  - Recalculer les nouvelles moyennes des k *clusters*. La moyenne μ_j du *cluster* K_j est calculée comme suite :

$$\mu_j = \frac{1}{|K_j|} \sum_{x_i \in K_j} x_i$$

$|K_j|$ est le nombre d'objets dans le *cluster* K_j

Jusqu'à ce qu'il n'y aucun changement

La méthode K-moyennes est simple à implémenter. Elle donne des bons résultats pour les *clusters* compacts et hyper-sphériques (lorsque la distance Euclidienne est utilisée, ce qui est le plus fréquent), elle ne dépend pas de l'ordre des données. De plus, sa complexité est relativement faible. En revanche, la méthode K-moyennes est sensible à l'initialisation des k premières moyennes et le résultat dépend de la valeur de k. De plus, en particulier en présence de données aberrantes, cette méthode peut converger vers un optimum local.

b) K-moyennes global

La méthode K-moyennes globales [Likas *et al.* 2003] est une variante de la méthode K-moyennes, itérative, où un *cluster* est ajouté à chaque itération. Autrement dit, pour partitionner les données en *clusters*, nous réalisons successivement la méthode K-moyennes avec des nombres des *clusters* $k_i = 1 \dots k^*$.

En détail, à l'étape i, nous appliquons la méthode K-moyennes avec le paramètre k = i. L'initialisation des k moyennes se fait de la manière suivante :

- Récupérer les i – 1 moyennes qui sont retournées à l'étape i – 1
- La i^{ième} moyenne est initialisée à l'objet x_h qui maximise la fonction b_h :

$$b_h = \sum_{j=1}^{N} \max\left(d_{i-1}^{j} - \left(x_n - x_j\right)^2, 0\right)$$

$$h = argmax(b_n)$$

où d_{i-1}^{j} est la distance carrée entre x_j et le moyenne le plus proche parmi les (i-1) moyennes initialisés

Après l'initialisation des k moyennes, l'algorithme K-moyennes est exécuté pour obtenir la solution avec k = i *clusters*.

La méthode K-moyennes globales est moins sensible aux conditions d'initialisation. Donc, cette méthode est plus efficace que la méthode K-moyennes, mais sa complexité est plus élevée. Le nombre de *clusters* pourrait être sélectionné automatiquement en arrêtant l'algorithme à la valeur k* qui donne les résultats acceptable suite selon une mesure interne.

c) K-médoïdes

La méthode K-médoïdes [Kaufman & Rousseeuw 1987] est similaire à la méthode K-moyennes, mais au lieu d'utiliser les moyennes comme représentants des *clusters*, la méthode K-médoïdes utilise des données bien choisies, pour éviter la sensibilité à l'égard des bruits et des valeurs aberrantes. Elle est applicable lorsque la moyenne n'est pas calculable.

L'algorithme PAM (Partitioning Around Medoids) [Theodoridis & Koutroumbas 2006] est la réalisation la plus courante de la méthode K-médoïdes. L'algorithme PAM est le suivant :

- Sélectionner au hasard k objets comme k médoïdes initiaux
- Répéter :
    - Attribuer chaque point au plus proche médoïde
    - Pour chaque paire {m,o} (m est le médoïde, o est un objet dans la base de données qui n'est pas le médoïde) : échanger le rôle de m et o (c'est-à-dire : o est le médoïde, m n'est plus le médoïde), puis, calculer la distance globale de cette solution
    - Sélectionner la solution dont la distance moyenne entre chaque médoïde et les données de son *cluster* est minimale et est la plus petite.

  Jusqu'à ce qu'il n'y a plus de changement dans les médoïdes.

La méthode K-médoïdes est couteuse en termes de la complexité, parce que le calcul de médoïde a une complexité quadratique : $O(k(N-k)^2)$

d) CLARA (*Clustering* LARge Applications)

L'idée de la méthode CLARA [Charrad & Ben Ahmed 2011] est d'appliquer la méthode K-médoïdes sur un petit échantillon des données, au lieu d'appliquer sur

toutes les données. La complexité est donc réduite. Après l'application de K-médoïdes sur l'échantillon des données, les autres objets qui ne sont pas dans l'échantillon seront attribués à *cluster* dont le médoïde est le plus proche.

L'algorithme CLARA est le suivant :

- Pour i = 1 à q :
    - Créer l'échantillon S en sélectionnant Ns objets au hasard dans l'ensemble des données E
    - Appliquer l'algorithme K-médoïdes sur l'échantillon S
    - Attribuer chaque point dans {E\S} au *cluster* dont le médoïde est le plus proche
    - Calculer la distance globale de cette solution
- Sélectionner la solution qui a la distance globale la plus petite

La complexité de l'algorithme CLARA est moins élevée que l'algorithme K-médoïdes : $O(kN_s^2 + k(N-k))$, l'algorithme CLARA est donc plus adapté aux grandes bases de données. Mais, le résultat dépend de l'échantillon sélectionné, et il peut converger vers un optimum local.

Conclusion: Nous avons présenté dans cette section les méthodes par partitionnement. Dans ces méthodes, le nombre de *clusters* est habituellement prédéfini. Parmi ces méthodes, la méthode K-moyennes est la plus connue et la plus utilisée grâce à sa simplicité et son efficacité. Les désavantages de ces méthodes ce sont que presque toutes les méthodes par partitionnement ne sont pas incrémentales, elles ne donnent pas une organisation hiérarchique de *clusters*. Or, cette organisation hiérarchique peut être intéressante dans notre application où nous pouvons chercher à diviser ou regrouper des clusters. Nous présentons dans la section suivante les méthodes hiérarchiques qui donnent une organisation hiérarchique de *clusters*.

B.2. Les méthodes hiérarchiques

Les méthodes hiérarchiques fournissent une décomposition hiérarchique des *clusters* aux sous-*clusters*, tandis que les méthodes de partitionnement donnent une organisation « plate » (linéairement séparable) de *clusters*. Quelques méthodes de ce type sont : CAH [Theodoridis & Koutroumbas 2006], BIRCH [Zhang *et al.* 1996], CURE [Theodoridis & Koutroumbas 2006], *etc.*

a) La méthode CAH (Classification Ascendant Hiérarchique)

La méthode CAH [Theodoridis & Koutroumbas 2006] est dite ascendante car elle commence à partir d'une situation où tous les objets sont seuls dans un *cluster*, puis, les objets sont regroupés en *clusters* de plus en plus grands. Plus précisément, à chaque étape, nous fusionnons deux *clusters*, réduisant ainsi le nombre de *clusters*. Les deux *clusters* choisis pour être fusionnés sont ceux qui les plus proches, en d'autres termes, ceux dont la dissimilarité est minimale.

L'algorithme est le suivant :

- Étape 1 : Attribuer chaque objet à un seul *cluster*. Donc, nous avons N *clusters* correspondent à N objets de la base de données.
- Étape 2 : Calculer les distances entre tous les paires de *clusters*
- Étape 3 : Fusionner deux *clusters* les plus proches
- Étape 4 : Répéter l'étape 2 et l'étape 3 jusqu'à ce qu'il n'y ait plus qu'un seul *cluster*

Il existe des différentes mesures de distance entre deux *clusters* K_i et K_j: $D(K_i, K_j)$:

- Le saut minimum : La distance entre deux *clusters* K_i et K_j est la distance minimale des distances entre un objet du *cluster* K_i et un autre objet du *cluster* K_j :

$$D(K_i, K_j) = \min_{x \in K_i, x' \in K_j} D(x, x')$$

- Le saut maximum : La distance entre deux *clusters* K_i et K_j est la distance maximale des distances entre un objet du *cluster* K_i et un autre objet du *cluster* K_j :

$$D(K_i, K_j) = \max_{x \in K_i, x' \in K_j} D(x, x')$$

- Le lien moyen : La distance entre deux *clusters* K_i et K_j est la distance moyenne des distances entre un objet du *cluster* K_i et un autre objet du *cluster* K_j :

$$D(K_i, K_j) = \frac{1}{|K_i||K_j|} \sum_{x \in K_i, x' \in K_j} D(x, x')$$

- Le lien des centroïdes : La distance entre deux *clusters* K_i et K_j est la distance entre deux centroïdes μ_i et μ_j de ces deux *clusters*
- La distance de Ward :

$$D(K_i, K_j) = \frac{|K_i||K_j|}{|K_i| + |K_j|}\left(\mu_{K_i} - \mu_{K_j}\right)^2$$

*o*ù μ_{K_i} et μ_{K_j} sont respectivement les centres des *clusters* K_i et K_j.

L'arbre construit par l'algorithme CAH est déterministe puisque cet algorithme n'a pas besoin d'initialisation. Mais, le désavantage de l'algorithme CAH, c'est que sa complexité (en temps et en mémoire vive) est élevée. De plus, il est sensible aux bruits et aux données aberrantes.

b) La méthode CURE (*Clustering* Using REpresentatives)

La méthode CURE [Theodoridis & Koutroumbas 2006] est similaire à la méthode CAH : tous les objets sont initialement seuls dans une classe, puis, nous fusionnons itérativement les paires de *clusters* les plus proches, réduisant ainsi le nombre de *clusters*. La seule différence entre ces deux méthodes, c'est la distance entre deux *clusters*. La méthode CURE calcule la distance entre deux *clusters* en se basant sur quelques objets représentatifs de ces deux *clusters* au lieu d'utiliser tous les objets. La complexité de la méthode CURE est donc réduite

Pour déterminer l'ensemble des objets représentatifs $K_i.rep$ d'un *cluster* K_i, d'abord, nous sélectionnons un sous-ensemble $tmpSet$ de $K_i.rep$ qui contient des objets de K_i bien dispersés. Puis, chaque objet $p \in tmpSet$ est rétréci vers la moyenne du *cluster* avant d'être ajouté à $K_i.rep$:

$$K_i.rep = K_i.rep \cup \{p + \alpha * (K_i.mean - p)\}$$

$K_i.mean$ est la moyenne de tous les objets dans K_i, α est un paramètre de fraction : $0 \leq \alpha \leq 1$

L'ensemble $tmpSet$ est construit comme suit : l'objet le plus loin de la moyenne μ_i est sélectionné comme le premier objet dispersé. Et puis, chaque nouvel objet dispersé est défini comme l'objet le plus loin de l'objet dispersé précédent.

En rétrécissant les objets vers la moyenne du *cluster*, nous pouvons réduire l'impact des valeurs aberrantes. Cependant, il est difficile de fixer le paramètre de fraction α.

Conclusion: Nous avons présenté dans cette section les méthodes hiérarchiques. L'avantage des méthodes hiérarchiques c'est qu'elles organisent les données sous une structure hiérarchique. Par conséquent, nous pouvons obtenir des nombres différents des *clusters* en considérant la structure hiérarchique à différents niveaux, et facilement diviser ou regrouper des clusters existants.

B.3. Les méthodes basées sur la densité

Les méthodes basées sur la densité visent à partitionner les données en basant sur la densité locale de ces données. Un groupe des données qui sont localement denses est considéré comme un *cluster*. Les objets dans les zones éparses sont considérés comme les bruits. Quelques méthodes dans ce type sont : DBSCAN [Ester *et al.* 1996], OPTICS [Ankerst *et al.* 1999], *etc.*

La méthode DBSCAN (Density Based Spatial *Clustering*) [Ester *et al.* 1996] est la méthode la plus populaire. Cette méthode est basée sur la densité locale des données pour identifier les sous-ensembles des données denses qui peuvent considérées comme des *clusters*.

Pour décrire l'algorithme, nous utilisons les termes suivants :

- Les $\epsilon - voisinages$ d'un objet p, noté $N_\epsilon(p)$, est un ensemble qui contient tous les objets q dont la distance entre q et p : $D(p,q) < \epsilon$
- $MinPts$ est une valeur constante utilisée pour déterminer les objets fondamentaux d'un *cluster*. Un objet est considéré comme un objet fondamental s'il y a au moins $MinPts$ points dans son $\epsilon - voisinage$
- *Densité-atteignable direct* : Un objet p est « densité-atteignable direct » à partir d'un objet q si q est un objet fondamental et $p \in N_\epsilon(q)$
- *Densité-atteignable :* Un objet p est « densité-atteignable » à partir d'un objet q s'il y a une chaîne d'objets : $p_1, p_2, \dots, p_n$ telle que $p_1 = q, p_n = p$ et $\forall i = 1 \dots n$, p_{i+1} est densité-atteignable direct à p_i
- *Densité-connecté* : Deux objets p et q sont « densité-connectés » s'il y a un objet o tel que p et q sont touts les deux densité-atteignables à partir de o.

L'algorithme DBSCAN est le suivant :

- Marquer tous les objets dans la base de données comme non-classifié
- Pour chaque objet non-classifié x_i
    - Si x_i est un objet fondamental :

- Attribuer x_i à un nouveau *cluster*
- Étendre le nouveau *cluster* de x_i afin qu'il contient tous les objets qui sont densité-atteignables à x_i

  - Si x_i n'est pas un objet fondamental, marquer x_i comme BRUIT

Différemment de la méthode K-moyennes par exemple, la méthode DBSCAN produit des *clusters* des formes variées. Le nombre de *clusters* n'a pas besoin d'être préalablement fixé. La complexité de cette méthode est moins élevée : $O(NlogN)$ (N est le nombre des objets dans la base de données). De plus, cette méthode est robuste aux valeurs aberrantes. En revanche, les paramètres ϵ et $MinPts$ sont difficiles à régler. De plus, l'algorithme DBSCAN n'arrive pas à identifier les *clusters* si la densité est variée d'un *cluster* à l'autre ou si les données sont trop . Donc, l'algorithme DBSCAN n'est pas adapté aux données de grande dimension.

B.4. Les méthodes basées sur modèles

Basées sur l'hypothèse que les données sont générées par un modèle donné, les méthodes basées sur modèle essaient de trouver le modèle des *clusters*, et puis, utilisent le modèle trouvé pour classifier les nouveaux objets. Les méthodes EM [Sundberg 1974; Dempster *et al.* 1977] et SOM [Kohonen 1982] sont des méthodes typiques de ce type.

1) La méthode EM (Expectation Maximization)

Supposons que les objets dans la base de données sont indépendants et qu'ils sont identiquement distribués selon un ensemble de k distributions Gaussiennes. La $j^{ième}$ distribution Gaussienne génère les objets du $j^{ième}$ *cluster*. La distribution Gaussienne d'un *cluster* K_j est caractérisée par sa moyenne μ_j et sa matrice de covariance Σ_j . L'objectif de l'algorithme EM [Dempster *et al.* 1977; Sundberg 1974] est d'optimiser les paramètres des distributions Gaussiennes en attribuant itérativement les données aux *clusters* pour maximiser la vraisemblance de ces paramètres.

L'algorithme EM est le suivant :

- Supposons que nous avons k distributions Gaussiennes, les paramètres initiaux du modèle sont :

$$\theta^0 = \{\mu_1^0, \mu_2^0, \dots, \mu_k^0, \Sigma_1^0, \Sigma_2^0, \dots, \Sigma_k^0, \alpha_1^0, \alpha_2^0, \dots, \alpha_k^0\}$$

*o*ù α_i représente la probabilité d'occurrence du *cluster* K_i

- Répéter jusqu'à convergence :
  - L'étape E : Ayant les paramètres du modèle, nous calculons, pour chaque objet x_i et chaque *cluster* K_j, la probabilité que x_i appartienne à K_j :

$$P\left(x_i \in K_j \middle| x_i, \theta^t\right) = \frac{P\left(x_i \middle| x_i \in K_j, \mu_j^t, \Sigma_j^t\right)\alpha_j^t}{\sum_{j=1}^{k} P\left(x_i \middle| x_i \in K_j, \mu_j^t, \Sigma_j^t\right)\alpha_j^t}$$

  - L'étape M : Mettre à jour les paramètres du modèle pour qu'ils maximisent sa vraisemblance :

$$\alpha^{t+1} = P\left(K_j\right) = \frac{1}{N}\sum_{i=1}^{N} P(x_i \in K_j | x_i, \theta^t)$$

$$\mu_j^{t+1} = \frac{\sum_{i=1}^{N} P\left(x_i \in K_j \middle| x_i, \theta^t\right) x_i}{\sum_{i=1}^{N} P(x_i \in K_j | x_i, \theta^t)}$$

$$\Sigma_j^{t+1} = \frac{\sum_{i=1}^{N} P\left(x_i \in K_j \middle| x_i, \theta^t\right)\left(x_i - \mu_j^{t+1}\right)\left(x_i - \mu_j^{t+1}\right)^T}{\sum_{i=1}^{N} P(x_i \in K_j | x_i, \theta^t)}$$

Lorsque les paramètres du modèle sont déterminés, l'algorithme EM attribue chaque objet x_i au plus vraisemblabe *cluster* K_j (dont la probabilité $P\left(x_i \in K_j \middle| x_i, \theta\right)$ est maximale)

L'algorithme EM est facile à implémenter. Il permet d'identifier les valeurs aberrantes. La complexité de cet algorithme est peu élevée : $O(Nk^2l)$ (l est le nombre d'itérations). Cependant, si les données de chaque *cluster* ne sont pas distribuées selon la distribution Gaussienne, les résultats sont souvent mauvais. De plus, l'algorithme EM peut converger à un optimum local.

2) La méthode SOM (Self Organizing Map)

La méthode SOM [Kohonen 1982] construit un réseau de neurone monocouche pour grouper les données similaires. Chaque neurone est associé à un vecteur de poids, représentant le centroïde d'un *cluster*. L'objectif de cette méthode est de « mapper » les données de grandes dimensions vers un plan en deux dimensions

en trouvant, pour chaque donnée d'entrée, le neurone associé au plus proche vecteur de poids.

L'algorithme SOM est le suivant :

- Initialiser les vecteurs de poids de tous les neurones (souvent au hasard)
- Répéter jusqu'à ce que les vecteurs de poids ne changent plus :
    - Sélectionner au hasard un objet dans la base de données et calculer la distance entre le vecteur d'entrée et tous les neurones dans la couche de sortie. Le neurone associé au vecteur de poids le plus proche est appelé le neurone gagnant.
    - Mettre à jour le vecteur de poids du neurone gagnant et des neurones dans son voisinage. Le vecteur de poids w_i du neurone i est mis à jour comme suit :

$$w_i(t+1) = w_i(t) + \alpha(t) + \phi_i(c)[x(t) - w_i(t)]$$

où t est l'indice de l'étape, c est l'indice du neurone gagnant pour le vecteur d'entrée $x(t)$, $\alpha(t)$ est le coefficient d'apprentissage, et $\phi_i(c)$ est le voisinage de c.

La fonction de voisinage $\phi_i(c)$ diminue avec la distance (dans le plan) entre le neurone gagnant et le neurone i de sorte que les vecteurs de poids des neurones qui sont loin du neurone gagnant sont moins modifiés que ceux des neurones qui sont proches du neurone gagnant.

La méthode SOM est incrémentale (les vecteurs de poids peuvent être mis à jour lorsque de nouvelles données arrivent). Cette méthode est adaptée aux grandes bases de données. La complexité de SOM est moins élevée : $O(Nkl)$ (N est le nombre d'objets dans la base de données, k est le nombre de neurones, l est le nombre d'itérations). Par contre, le résultat dépend des valeurs d'initialisation et de la fonction de voisinage.

Annexe C. Consensus *clustering*

Dans l'annexe B, nous avons présenté quelques-unes des méthodes de *clustering* selon une typologie fixée. Il y a des inconvénients pour toutes les méthodes de *clustering* existantes. Un de ces inconvénients, fréquemment observé, est la sensibilité aux paramètres initiaux. Un autre inconvénient est que souvent, ces algorithmes nécessitent de connaitre le nombre k de clusters. De plus, chaque type de méthode (par partitionnement hiérarchique, *etc.*) a ses propres avantages et

inconvénients, et chacun donne souvent des résultats sensiblement différents. Pour ces raisons, il est utile de combiner des différents *clusterings* (qui sont déjà obtenus par quelques algorithmes de *clustering*) pour retrouver un seul *clustering* de bonne qualité. C'est le but du consensus *clustering*. Le résultat du consensus *clustering* est plus stable et est moins sensible aux conditions initiales que chacune des méthodes prises individuellement. De plus, il peut permettre de choisir automatiquement le nombre de clusters.

Pour faire le consensus *clustering*, une approche possible est de choisir le *clustering* qui partage le plus d'information avec tous les autres *clusterings*. Une mesure est donc nécessaire pour mesurer la quantité d'information partagée entre les *clusterings*. De nombreuses mesures pour comparer les *clusterings* ont été proposées. Nous pouvons définir une taxonomie de ces mesures qui distingue 3 catégories :

- *Les mesures basées sur le comptage de paires* : Ces mesures sont construites sur le comptage des paires des objets sur lesquels les 2 *clusterings* sont en accord ou en désaccord.
- *Les mesures basées sur l'appariement des clusters* : Ces mesures sont basées sur la recherche des correspondances entre les *clusters* dans les 2 *clusterings*.
- *Les mesures basées sur la théorie de l'information* : Ces mesures sont construites sur les concepts fondamentaux de la théorie d'information.

Dans les sous-sections suivantes, nous présentons quelques-unes des méthodes de ces 3 catégories. Dans notre présentation, nous utilisons les notions suivantes :

- S est l'ensemble de N objets que l'on cherche à regrouper en clusters.
- U est un *clustering* sur S qui partitionne S en *clusters* (non-recouvrement) : $\{U_1, U_2, \dots, U_R\}$ où $\bigcup_{i=1}^{R} U_i = S$ et $U_i \cap U_j = \emptyset$ pour $i \neq j$
- V est un *clustering* sur S qui partitionne S en *clusters* (non-recouvrement) : $\{V_1, V_2, \dots, V_C\}$ où $\bigcup_{i=1}^{C} V_i = S$ et $V_i \cap V_j = \emptyset$ pour $i \neq j$
- $n_{ij} (i = 1..R, j = 1..C)$ est le nombre d'objets communs aux *clusters* U_i et V_j
- $a_i = \sum_{j=1}^{C} n_{ij}$ $(i = 1..R)$
- $b_j = \sum_{i=1}^{R} n_{ij}$ $(j = 1..C)$
- N_{11} est le nombre de paires d'objets qui sont dans le même *cluster* dans les deux *clusterings* U et V.
- N_{00} est le nombre de paires d'objets qui sont dans différents *clusters* dans les deux *clusterings* U et V.
- N_{01} est le nombre de paires d'objets qui sont dans le même *cluster* du U mais dans différents *clusters* du V.

- N_{10} est le nombre de paires d'objets qui sont dans différents *clusters* du U mais dans le même *cluster* du V.

C.1. Les mesures basées sur le comptage de paires

Ces mesures sont construites sur le comptage des paires des objets sur lesquels les 2 *clusterings* sont en accord ou en désaccord. Albatineh *et al.* [Albatineh *et al.* 2006] font une liste complète des 22 différentes mesures de ce type. Parmi ces 22 mesures, la mesure RI et la mesure ARI sont les mesures les plus connues et largement utilisées.

Rand *et al.* [Rand 1971] proposent la mesure Rand Index (RI), définie comme suite :

$$RI(U,V) = \frac{N_{00} + N_{11}}{\binom{N}{2}}$$

La valeur de base de la mesure RI est élevée et n'est pas constante. Hubert et Arabie [Hubert & Arabie 1985] proposent donc la version ajustée de la mesure RI, appelée ARI (Adjusted Rand Index) :

$$ARI(U,V) = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00} + N_{01})(N_{01} + N_{11}) + (N_{00} + N_{10})(N_{10} + N_{11})}$$

C.2. Les mesures basées sur l'appariement des clusters

Ces mesures sont basées sur la recherche de correspondances entre *clusters* dans les 2 *clusterings*. Quelques mesures de ce type sont discutées dans Meila [Meila 2007]. Les problèmes de ces mesures sont liées :

- Au fait que le nombre de *clusters* dans les deux *clusterings* peut être différent.
- Au fait que même si le nombre de *clusters* dans les deux *clusterings* est le même, l'ensemble des données non-appariées de chaque paire appariée de *clusters* n'est pas prise en considération.

C.3. Les mesures basées sur la théorie de l'information

Ces mesures sont construites sur les concepts fondamentaux de la théorie de l'information. Étant donnés 2 *clusterings* U et V, leurs entropies $H(U)$, $H(V)$, leurs entropies conjointes $H(U,V)$, $H(V,U)$, leurs entropies conditionnelles $H(U|V)$,$H(V|U)$ et l'information mutuelle $I(U,V)$ sont définies comme suit :

$$H(U) = -\sum_{i=1}^{R} \frac{a_i}{N} \log \frac{a_i}{N}$$

$$H(V) = -\sum_{j=1}^{C} \frac{b_j}{N} \log \frac{b_j}{N}$$

$$H(U,V) = H(V,U) = -\sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}}{N}$$

$$H(U|V) = -\sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}/N}{b_j/N}$$

$$H(V|U) = -\sum_{j=1}^{C} \sum_{i=1}^{R} \frac{n_{ij}}{N} \log \frac{n_{ij}/N}{a_i/N}$$

$$I(U,V) = \sum_{i=1}^{R} \sum_{j=1}^{C} \frac{n_{ij}}{N} \log \frac{n_{ij}/N}{a_i b_j/N^2}$$

Supposons que nous devons transmettre toutes les étiquettes des *clusters* dans U sur un canal de communication. Supposons que V est mis à la disposition du récepteur. Donc, $H(U)$ peut être interprété comme le montant moyen d'information nécessaire pour coder les étiquettes des *clusters* dans U. $H(U|V)$ indique le montant moyen d'information nécessaire pour transmettre chaque étiquette dans U si V est déjà connu.

Nous sommes intéressés à la façon de voir combien la connaissance de V nous aide à réduire le nombre d'informations nécessaires pour coder U. Ceci peut être quantifié en termes de l'information mutuelle. Il nous dit combien le fait de connaître un de ces *clusterings* réduit l'incertitude sur l'autre :

$$I(U,V) = H(U) - H(U|V) = H(V) - H(V|U)$$

Le Tableau 6.1 montre quelques formules des mesures basées sur la théorie de l'information dans la littérature :

Nom de la mesure	***Formule***	***Référence***
MI (Mutual Information)	$I(U,V)$	[Banerjee & Dhillon 2005]
MI normalisé (NMI)		
NMI_{joint}	$\frac{I(U,V)}{H(U,V)}$	[Yao 2003]
NMI_{max}	$\frac{I(U,V)}{\max\big(H(U),H(V)\big)}$	[Kvalseth 1987]
NMI_{sum}	$\frac{2I(U,V)}{H(U)+H(V)}$	[Kvalseth 1987]
NMI_{sqrt}	$\frac{I(U,V)}{\sqrt{H(U)H(V)}}$	[Strehl & Ghosh 2003]
NMI_{min}	$\frac{I(U,V)}{\min\big(H(U),H(V)\big)}$	[Kvalseth 1987] [Liu *et al.* 2008]
MI ajusté (AMI)		
AMI_{max}	$\frac{I(U,V)-E\big(I(U,V)\big)}{\max\big(H(U),H(V)\big)-E\big(I(U,V)\big)}$	[Vinh 2010a]
AMI_{sum}	$\frac{I(U,V)-E\big(I(U,V)\big)}{\frac{1}{2}\Big(\big(H(U)+H(V)\big)-E\big(I(U,V)\big)\Big)}$	[Vinh 2010b]

AMI_{sqrt}	$\frac{I(U,V) - E\big(I(U,V)\big)}{\sqrt{H(U)H(V)} - E\big(I(U,V)\big)}$	[Vinh 2010b]
AMI_{min}	$\frac{I(U,V) - E\big(I(U,V)\big)}{\min\big(H(U),H(V)\big) - E\big(I(U,V)\big)}$	[Vinh 2010b]

Tableau 6.1 : Quelques formules des mesures basées sur la théorie de l'information

Dans les formules des mesures AMI, la valeur $E\big(I(U,V)\big)$ est calculée comme suit :

$$E\big(I(U,V)\big) = \sum_{i=1}^{R}\sum_{j=1}^{C}\sum_{n_{ij}=\max(a_i+b_j-N,0)}^{\min(a_i,b_j)} \frac{n_{ij}}{N}\log\left(\frac{N.n_{ij}}{a_i b_j}\right)\frac{a_i!\,b_j!\,(N-a_i)!\,(N-b_j)!}{N!\,n_{ij}!\,(a_i-n_{ij})!\,(b_j-n_{ij})!\,(N-a_i-b_j+n_{ij})!}$$

C.4. Les propriétés des mesures de clustering

Dans cette section, nous présentons quelques propriétés désirables d'une mesure de *clustering* :

- *Métrique* : Cette propriété exige que la mesure satisfasse les propriétés d'une vraie métrique : définie positive, symétrique et vérifiant l'inégalité triangulaire. Nous pouvons démontrer que :
  - Les mesures RI et ARI ne sont pas métriques
  - Les mesures NMI_{sum}, NMI_{sqrt}, NMI_{min}, AMI_{max}, AMI_{sum}, AMI_{sqrt}, AMI_{min} ne sont pas métriques
  - Les mesures NMI_{joint} et NMI_{max} sont métriques.
- *Normalisation* : Cette propriété exige que les valeurs de la mesure se situe dans un intervalle fixe, par exemple [0,1].
- *Base constante* : Pour une mesure, l'espérance des mesures entre des paires de *clusterings* indépendants devrait être une constante. Idéalement, elle devrait être nulle, indiquant que la similarité entre 2 *clusterings* indépendants est zéro. Nous pouvons démontrer que :
  - La mesure ARI est de base constante
  - Les mesures AMI_{max}, AMI_{sum}, AMI_{sqrt}, AMI_{min} sont de base constante

- Les mesures NMI_{joint}, NMI_{max}, NMI_{sum}, NMI_{sqrt}, NMI_{min} ne sont pas de base constante. Mais, lorsque le nombre d'objets dans la base de données est beaucoup supérieur que le nombre de *clusters*, la valeur moyenne de ces mesures est assez proche de zéro [Vinh 2010b].

Références

Albatineh, A.N.Niewiadomska-Bugaj, M.and Mihalko, D. 2006. On Similarity Indices and Correction for Chance Agreement. *Journal of Classification*, 23, pp.301–313.

Aldavert, D. et al. 2013. Integrating Visual and Textual Cues for Query by String Word Spotting. In *International Conference on Document Analysis and Recognition*.

Ankerst, M. et al. 1999. OPTICS: ordering points to identify the *clustering* structure. *ACM SIGMOD Record*, 28, pp.49–60. Available at: http://portal.acm.org/citation.cfm?id=304187.

Banerjee, A. and Dhillon, I. 2005. *Clustering* on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*, pp.1345–1382. Available at: http://machinelearning.wustl.edu/mlpapers/paper_files/BanerjeeDGS05.pdf.

Bauckhage, C. and Tsotsos, J.K. 2005. Bounding box splitting for robust shape classification. *IEEE International Conference on Image Processing 2005*, 2.

Belongie, S.Malik, J.and Puzicha, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24.

Bianne-Bernard, A.-L. et al. 2011. Dynamic and Contextual Information in HMM Modeling for Handwritten Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33, pp.2066–2080.

Bunke, H. and Shearer, K. 1998. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4), pp.255–259.

Celebi, M.E. and Aslandogan, Y.A. 2005. A comparative study of three moment-based shape descriptors. *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, 1.

Cron, G. 2012. *Rapport interne de la Bibliothèque Nationale de France*,

Charrad, M. and Ben Ahmed, M. 2011. Simultaneous *Clustering*: A Survey. *Pattern Recognition and Machine Intelligence*, 6744, pp.370–375. Available at: http://www.springerlink.com/content/j28g34r48x612t64/.

Dempster, A.P.Laird, N.M.and Rubin, D.B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B Methodological*, 39, pp.1–38. Available at: http://www.jstor.org/stable/2984875.

España-Boquera, S. et al. 2011. Improving offline handwritten text recognition with hybrid HMM/ANN models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33, pp.767–779.

Ester, M. et al. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Computer*, 1996, pp.226–231. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Density-Based+Algorithm+for+Discovering+Clusters+in+Large+Spatial+Databases+with+Noise#0.

Fischer, A. et al. 2012. Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*, 33, pp.934–942.

Frakes, W.B. and Baeza-Yates, R. 1992. *Information Retrieval: Data Structures and Algorithms*, Available at: http://www.citeulike.org/group/328/article/308697.

Graves, A.Fernández, S.and Liwicki, M. 2008. Unconstrained online handwriting recognition with recurrent neural networks. *Advances in Neural …*, 20, pp.1–8. Available at: http://www.dfki.uni-kl.de/~liwicki/pdf/GrFeSchLiBu07-01.pdf.

Hart, P.E.Nilsson, N.J.and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4.

Homer, E.L. 2000. Extraction of *strokes* in handwritten characters. *Pattern Recognition*, 33(April 1999), pp.1147–1160.

Hu, J.Gek Lim, S.and Brown, M.K. 2000. Writer independent on-line handwriting recognition using an HMM approach. *Pattern Recognition*, 33, pp.133–147.

Hu, J.H.J.Brown, M.K.and Turin, W. 1996. HMM based online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18.

Huang, T.H.T. and Yasuhara, M. 1995. A total *stroke* SLALOM method for searching for the optimal drawing order of off-line handwriting. *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*, 3.

Hubert, L. and Arabie, P. 1985. Comparing partitions. *Journal of Classification*, 2, pp.193–218.

Impedovo, S.Ottaviano, L.and Occhinegro, S. 1991. Optical Character Recognition - a survey. *International Journal of Pattern Recognition and Artificial Intelligence*, pp.1–24.

Jager, S. 1996. Recovering writing traces in off-line handwriting recognition: Using a global optimization technique. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. pp. 150–154.

Kato, Y. and Yasuhara, M. 2000. Recovery of drawing order from single-*stroke* handwriting images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22, pp.938–949.

Kaufman, L. and Rousseeuw, P. 1987. *Clustering by means of Medoids*,

Kavallieratou, E.Fakotakis, N.and Kokkinakis, G. 2002. An unconstrained handwriting recognition system. *Journal on Doucment Analysis and Recognition*, 4(4), pp.226–242.

Keaton, P.Greenspan, H.and Goodman, R. 1997. Keyword spotting for cursive document retrieval. *Proceedings Workshop on Document Image Analysis (DIA'97)*.

Kieu, V.C. et al. 2013. Semi-synthetic Document Image Generation Using Texture Mapping on Scanned 3D Document Shapes. In *International Conference on Document Analysis and Recognition*. pp. 489–493.

Kim, I.-J.K.I.-J. and Kim, J.-H.K.J.-H. 2003. Statistical character structure modeling and its application to handwritten Chinese character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25,

pp.1422–1436. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240117.

Kohonen, T. 1982. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43, pp.59–69. Available at: http://dx.doi.org/10.1007/BF00337288.

Konidaris, T.Gatos, B.and Ntzios, K. 2007. Keyword-guided word spotting in historical printed documents using synthetic data and user feedback. *International Journal on Document Analysis and Recognition*, pp.167–177.

Kvalseth, T.O. 1987. Entropy and Correlation: Some Comments. *IEEE Transactions on Systems, Man, and Cybernetics*, 17, pp.517–519.

Lallican, P.M. 1997. A kalman approach for offline *stroke* order recovering. In *International Conference on Document Analysis and Recognition*. pp. 519–522.

Lee, C. and Wu, B. 1998. A chinese character *stroke* extraction algorithm based on contour information. *Pattern Recognition*, 31, pp.651–663.

Leydier, Y. et al. 2009. Towards an omnilingual word retrieval system for ancient manuscripts. *Pattern Recognition*, 42, pp.2089–2105.

Leydier, Y.Lebourgeois, F.and Emptoz, H. 2007. Text search for medieval manuscript images. *Pattern Recognition*, 40, pp.3552–3567.

Liang, Y.Fairhust, M.and Guest, R. 2012. A synthesisted word approach to word retrieval in handwritten documents. *Pattern Recognition*, pp.4225–4236.

Likas, A.Vlassis, N.and Verbeek, J.J. 2003. The global k-means *clustering* algorithm. *Pattern Recognition*, 36(2), pp.451–461.

Liu, K.Huang, Y.S.and Suen, C.Y. 1999. Identification of fork points on the skeletons of handwritten Chinese characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21, pp.1095–1100.

Liu, K.L.K.Huang, Y.S.and Suen, C.Y. 1997. Robust *stroke* segmentation method for handwritten Chinese character recognition. *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, 1.

Liu, Z.Guo, Z.and Tan, M. 2008. Constructing tumor progression pathways and biomarker discovery with fuzzy kernel kmeans and DNA methylation data. *Cancer informatics*, 6, pp.1–7.

Liwicki, M. and Bunke, H. 2006. HMM Based On-Line Recognition of Handwritten Whiteboard Notes. In *Proc. 10th Int. Workshop on Frontiers in Handwriting Recognition*. pp. 595–599.

MacQueen, J.B. 1967. Kmeans Some Methods for classification and Analysis of Multivariate Observations. *5th Berkeley Symposium on Mathematical Statistics and Probability 1967*, 1, pp.281–297. Available at: http://projecteuclid.org/euclid.bsmsp/1200512992.

Manmatha, R. et al. 2012. A Novel Word Spotting Method Based on Recurrent Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34, pp.211–224.

Manmatha, R.Han, C.and Riseman, E. 1996. Word spotting: A new approach to indexing handwriting. In *Computer Vision and Pattern Recognition*.

Marinai, S.Marino, E.and Soda, G. 2006. Font adaptive word indexing of modern printed documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28, pp.1187–1199.

Meila, M. 2007. Comparing clusterings-an information based distance. *Journal of Multivariate Analysis*, 98, pp.873–895.

N., A. et al. 2007. Shape retrieval using triangle-area representation and dynamic space warping. *Pattern Recognition*, 40(7), pp.1911–1920.

Nakajima, Y. et al. 1999. Global methods for *stroke* segmentation. *International Journal on Document Analysis and Recognition*, 2(1), p.19. Available at: http://link.springer.com/10.1007/s100320050032.

Perronnin, F. et al. 2008. Local gradient histogram features for word spotting in unconstrained handwritten documents. *International Conference on Frontiers in Handwriting Recognition*.

Plamondon, R. and Privitera, C.M. 1999. The segmentation of cursive handwriting: an approach based on off-line recovery of the motor-temporal information. *IEEE Transactions on Image Processing*, 8, pp.80–91. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18262867.

Plamondon, R. and Srihari, S.N. 2000. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22.

Qiao, Y.Nishiara, M.and Yasuhara, M. 2006. A framework toward restoration of writing order from single-stroked handwriting image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28, pp.1724–1737. Available at: http://www.ncbi.nlm.nih.gov/pubmed/17063679.

Qiao, Y.Q.Y. and Yasuhara, M. 2004. Recovering dynamic information from static handwritten images. *Ninth International Workshop on Frontiers in Handwriting Recognition*.

Rand, W.M. 1971. Objective Criteria for the Evaluation of *Clustering* Methods. *Journal of the American Statistical Association*, 66, pp.846–850. Available at: http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356\npapers3://publication/doi/10.2307/2284239?ref=search-gateway:4f9c0eebf0704e55403232bdc329541e.

Rath, T.M. and Manmatha, R. 2003. Word image matching using dynamic time warping. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2003 Proceedings*, 2, pp.II–521–II–527.

Raveaux, R.Burie, J.C.and Ogier, J.M. 2010. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5), pp.394–406.

Rice, S.Kanai, J.and Nartker, T.A. 1992. *Report on the Accuracy of OCR Devices*,

Riesen, K. and Bunke, H. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27, pp.950–959.

Rocchio, J. 1971. *Relevance feedback in information retrieval*,

Rodríguez-Serrano, J.A. and Perronnin, F. 2009. Handwritten word-spotting using hidden Markov models and universal vocabularies. *Pattern Recognition*, 42, pp.2106–2116.

Rothfeder, J.L.Feng, S.and Rath, T.M. 2003. Using Corner Feature Correspondences to Rank Word Images by Similarity. *2003 Conference on Computer Vision and Pattern Recognition Workshop*, 3.

Rousseeuw, P. 1987. Silhouettes: A graphical aid to the interpretation and validation of *cluster* analysis. *Journal of Computational and Applied Mathematics*, 20, pp.53–65. Available at: http://linkinghub.elsevier.com/retrieve/pii/0377042787901257.

Rusinol, M. and Llados, J. 2013. Boosting the Handwritten Word Spotting Experience By Including the user in the loop. *Pattern Recognition*.

Senior, A.W. and Fallside, F. 1993. An off-line cursive script recognition system using recurrent error propagation networks. In *International Workshop on Frontiers in Handwriting Recognition*. pp. 132–141.

Simon, J.-C. and Baret, O. 1992. Regularities and Singularities in Line Pictures. In *Structured Document Analysis*. pp. 261–281.

Strehl, A. and Ghosh, J. 2003. *Cluster* ensembles---a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3, pp.583–617. Available at: http://dl.acm.org/citation.cfm?id=944935.

Su, Z.Cao, Z.and Wang, Y. 2009. *Stroke* extraction based on ambiguous zone detection: a preprocessing step to recover dynamic information from handwritten Chinese characters. *International Journal on Document Analysis and Recognition (IJDAR)*, 12(2), pp.109–121. Available at: http://link.springer.com/10.1007/s10032-009-0085-9 [Accessed October 10, 2013].

Suen, C.Y. et al. 1993. Building a new generation of handwriting recognition systems. *Pattern Recognition Letters*, 14, pp.303–315.

Sundberg, R. 1974. Maximum likelihood theory for incomplete data from an exponential family. *Scandinavian Journal of Statistics*, 1, p.49/58. Available at: http://www.jstor.org/discover/10.2307/4615553?uid=3738880&uid=2&uid=4&sid=56307160993.

Teh, C.-H. and Chin, R.T. 1989. On the detection of dominant points on digital curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11.

Theodoridis, S. and Koutroumbas, K. 2006. *Pattern Recognition, Third Edition*, Available at: http://www.amazon.com/Pattern-Recognition-Edition-Sergios-Theodoridis/dp/1597492728.

Vinh, N.X. 2010a. Information Theoretic Measures for Clusterings Comparison : Variants , Properties , Normalization and Correction for Chance. *Journal of Machine Learning Research*, 11, pp.2837–2854. Available at: http://dl.acm.org/citation.cfm?id=1953024.

Vinh, N.X. 2010b. Information Theoretic Measures for Clusterings Comparison : Variants , Properties , Normalization and Correction for Chance. *Journal of Machine Learning Research*, 11, pp.2837–2854.

Wang, P. 2014. Novel learning-free word spotting approach based on graph representation. In *IAPR International Workshop on Document Analysis Systems*.

Welch, L.R. 2003. Hidden Markov Models and the Baum-Welch Algorithm. *IEEE Information Theory Society Newsletter*, 53, pp.1,10–13. Available at: http://www.itsoc.org/publications/nltr/it_dec_03final.pdf.

Wolf, C.Jolion, J.-M.and Chassaing, F. 2002. Text localization, enhancement and binarization in multimedia documents. *Object recognition supported by user interaction for service robots*, 2.

Yang, M.Kpalma, K.and Ronsin, J. 2008. A survey of shape feature extraction techniques. *Pattern recognition*, 2008, pp.43–90. Available at: citeulike-article-id:10157537\nhttp://hal.archives-ouvertes.fr/docs/00/44/60/37/PDF/ARS-Journal-SurveyPatternRecognition.pdf\nhttp://hal.archives-ouvertes.fr/hal-00446037/.

Yao, Y. 2003. Information theoretic measures for knowledge discovery and data mining. *Entropy Measures, Maximum Entropy Principle and Emerging Applications*, pp.115–136.

Zhang, T.Ramakrishnan, R.and Livny, M. 1996. BIRCH: an efficient data *clustering* method for very large databases. *Computer*, 1, pp.103–114. Available at: http://portal.acm.org/citation.cfm?id=233324.